Isosurface Extraction and Haptic Rendering of Volumetric Data

Kwong-Wai, Chen

A Thesis Submitted in Partial Fulfilment
of the Requirements for the Degree of
Master of Philosophy
in
Computer Science & Engineering

August 2000

Isosurface Extraction and Haptic Rendering of Volumetric Data

submitted by

Kwong-Wai, Chen

for the degree of Master of Philosophy
at the Chinese University of Hong Kong

Abstract

Many objects or natural phenomena are often described as volumetric models rather than geometric models in computer graphics. Therefore, researchers need appropriate methods to examine volumetric data. This thesis concerns with techniques of displaying volumetric data using both visual and haptic methods.

We first propose a new isosurface extraction algorithm, *multi-body surface extraction*, which efficiently generates a usable multi-body structure in a single processing pass through the volumetric data. Surface-based rendering techniques, particularly those that approximate an object surface embedded in volumetric data by a polygonal mesh, are widely used in volume visualization. However, if the problem is to extract the surfaces of multiple objects within the volumetric data, current techniques do not give a complete and consistent structure. Our algorithm does not generate the three-dimensional surface directly. Instead, we first compute iso-points, then iso-lines and finally the isosurfaces. By constructing the surface processing from low geometric dimensions upward to high dimensions, we can create a precisely multi-body surface delineating different types of objects.

Using volume visualization techniques, we can display meaningful information in volumetric data using interactive computer. Haptic rendering of volumetric data adds a new modality to volume visualization that is well-suited for presenting complex attributes of local region. However, many traditional haptic rendering methods are proposed to compute realistic interaction force with virtual environments consist of geometric primitives only. Recently, the benefits of haptic rendering of volumetric data have been recognized, but this area of research has not yet been fully explored. Our research is to develop direct haptic rendering techniques of volumetric data that can be used to build multi-sensory volume visualization systems.

論文題目：體數據的等值面抽取及觸感繪製
作者　　：陳罡鞹
學院　　：香港中文大學
學部　　：計算機科學與工程
修讀學位：哲學碩士

摘要

在計算機繪圖學，有許多的物件或者自然現象並非是以幾何模型所表示，而是形造成體數據 (Volumetric Data)。因此，研究人員需要各種的方法來檢查他們的數據。這篇論文主要探討了如何將體數據顯示於視覺上和觸覺上的技術。

首先，我們會提出一個全新的等值面抽取運算法 (Isosurface Extraction Algorithm)，稱爲 Multi-body Surface Extraction。其特點是能夠有效地在單一次的處理步驟後，便從一體數據中生成一個可用的多物體組織 (Multi-body Structure)。基於表面的繪製技術，特別是那些利用一個近似的多邊形網來表示一個存在於體數據內的物件表面，被廣泛地使用於體數據可視化 (Volume Visualization)。但是，如果問題是要抽取多個物體之間的表面，使用現時的技術並不能夠得到一個完整且一致的組織。我們所提出的運算法並非直接地建造一個三維的等值面，而是首先計算出等值點，然後等值線，最後才產生成等值面。這樣的運算方法有利於我們從體數據中抽取出一個準確吻合的、存在於多個物體之間的表面組織。

隨著體數據可視化技術的發展，我們可以把體數據內有用的訊息顯示在互動的電腦圖形和圖象上。體數據的觸感繪製 (Haptic Rendering)，可說是在體數據可視化上加上一個新的形態，它更能表示出局部區域的複雜屬性。雖然傳統的觸感繪製方法能夠計算出與虛擬物件接觸時的逼真的相互作用力，但是它們只能被應用於由幾何基礎模型所構成的虛擬環境中。我們的研究是要發展出體數據的直接觸覺繪感技術，從而被應用於建造多感官 (Multi-sensory) 的體數據視像系統。

Acknowledgments

First, I would like to thank my advisors, Pheng-Ann Heng and Hanqiu Sun, who spent time and effort throughout my graduate studies. For the past two years, they gave me invaluable suggestions, encouragement and guidance on my research and my thesis. I must also thank Andrew J. Hanson, Kin-Hong Wong, and Yiu-Sang Moon who reviewed the initial draft of this thesis and provided constructive comments.

Second, I would like to thank Tim Poston of the Department of Mathematics, National University of Singapore, for his insightful opinions and comments during our collaboration in the development of the *multi-body surface extraction* algorithm. I must also thank Tien-Tsin Wong, Michael Ping-Fu Fung, Kevin Chun-Ho Wong, Philip Chi-Wing Fu and Terence Kar-Ki Chan, who gave me lots of ideas and suggestions during different phrases of my research.

Third, I would like to thank my fellow colleagues, who have been resourceful for providing me technical and non-technical information, fun, moral support and other help in these two years. They are (not in order) Clara Tsui-Ying Law, Po-Shan Kam, Wai-Ching Wong, Wai-Chui Wong, Wing-Kai Lam, Sam Ka-Po Ma, Anthony Hing-Wing Chan, Steve Ka-Lung Chong, Yim-Pan Chui, Steve Yin-Hung Kuo, Chris Ting-On Lee, Anthony Yin-Pong Wong, Alex Wing-Keung Sim, and David, Yuk-Ming Chan. I must also thank the Department of Computer Science & Engineering, CUHK. It provides the best equipment and office environment required for high quality research to our students.

Finally, I would like to thank my parents, Kuo-Hui Chen and Lai-Lai Lam, my grandmother, Sau-Min Wong, my younger sister Ying-Ying Chen, who really make it possible for me to come this far. They encouraged me over the years, and supported my decision to continue my education.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Volumetric data can be obtained in many areas, by sampling, simulation, or modeling techniques. In medicine and industry, a sequence of 2D slices obtained from Magnetic Resonance Imaging (MRI) or Computed Tomography (CT) is 3D reconstructed into a volumetric model and visualized for non-destructive inspection. Similarly, confocal microscopes produce volumetric data that is visualized to study the morphology of biological structures. In computational fluid dynamics, the results of simulation are often modeled as volumetric data and visualized for analysis and verification.

Volume visualization is the technique used to display the information inside volumetric data using interactive graphics and imaging. While it has been proven quite effective for most application, it remains worthwhile to investigate the benefit of augmenting these visualization methods with information obtained through other sensory channels. In particular, our sense of touch, in combination with our kinesthetic sense, is capable of supplying a large amount of information about the structure, location, and material properties of objects [38, 20, 12, 46]. Recognizing this, in this thesis, we proposed techniques not only on visualization, but also haptic interaction of volumetric data, especially at regions of constant values.

In this chapter, we begin with an introduction to volumetric data in **Section 1.1**. **Section 1.2** covers briefly the volume visualization methods. We state the contributions of this thesis in **Section 1.3** and the outline in **Section 1.4**.

1.1 Volumetric Data

Volumetric data is typically a set of samples (x, y, z, v), representing the value v of some property of the data, at a 3D location (x, y, z). The value v may be simply a

scalar representing some measurable property of the data, including, for example, color, density, heat or pressure. The value v may even be a vector, representing, for example, velocity at each location. In general, the samples may be taken at purely random locations. Depending on how the samples are connected to form a grid structure, there are two classes of volumetric data: *structured* or *unstructured.*

Structured data has two components - a logical organization of the samples into a 3D array (called also *volume buffer*, *cubic frame buffer*, *3D raster*), and a mapping of each sample into the physical domain. The logical organization of the samples is called *computational space*, while the organization in physical domain is called *physical space.* The connectivity of the samples is implicitly identical in both logical and physical organization for structured data. There are four main types of structured data: *cartesian*, *regular*, *rectilinear* and *curvilinear.* While all four data types have an underlying logical organization of a regular 3D array, they vary in the physical mapping. In *cartesian* data, the samples are orthogonal axis-aligned with a constant and identical spacing along each axis. There is an implied physical mapping identical to its logical organization. When the spacing between samples along each axis is a constant, but there maybe three different spacing constants for the three axes, the data is *regular.* Samples of *rectilinear* data are orthogonal axis-aligned, but with non-constant spacing along each axis. The most complex structured data is *curvilinear*, where the samples are non-aligned with variable spacing. Therefore, each sample in logical organization has an explicit coordinate specifying its location in the physical domain. *Unstructured* data is a set of connected samples in space, whose connectivity has to be specified explicitly. Unlike structured data, unstructured data is not based on logical organization of array, but instead upon groups of cells. These cells can be of an arbitrary shape such as tetrahedra, hexahedra, or prism. In most cases, the volumetric data are either obtained as or re-sampled into cartesian structured grid. There are several advantages of cartesian data. First of all, memory consumption is optimized as the connectivity and physical mapping are implicit in the structure. In addition, visualization techniques can readily be used as data arrays map straightforwardly into pixel(2D) or voxel(3D) arrays.

In cartesian structured data, the 3D array (denoted as S, and $S(x, y, z)$ is the sample value v at grid point (x, y, z)), used to stored the samples defined only at grid position. Therefore, a function $f(x, y, z)$ is defined over R^3 in order to describe the value at any continuous location. The function $f(x, y, z) = S(x, y, z)$, if (x, y, z) is a grid location, otherwise $f(x, y, z)$ approximates the sample value at a location (x, y, z) by applying some interpolation function to S. There are many possible interpolation functions. The simplest interpolation function is known as *zero-order interpolation*,

which is actually just a nearest-neighbor function. The value at any location in R^3 is simply the value of the closest sample to that location. With this interpolation method, there is a region of constant value around each sample in S. Since the samples in S are regularly spaced, each region is a uniform sized cube (known as a *voxel*). Higher-order interpolation functions can also be used to define $f(x, y, z)$ between sample points. One common interpolation function is a piecewise function known as *first-order interpolation*, or *trilinear interpolation*. With this interpolation function, the value is assumed to vary linearly along each axis. To cite an example, consider the point P lying at location (x_p, y_p, z_p), within a cube of the 3D gird (known as a *cell*). The extreme grid points of the cube are $P_l = (x_l, y_l, z_l)$ and $P_h = (x_h, y_h, z_h)$. Then the value of P is interpolated, (1) along the four vertical edges of the cube to find the value of the four points (a_1, a_2, a_3, a_4) which form a square at $y = y_p$; (2) along two edges of the square to find the value of the two points (b_1, b_2) that form a line at $y = y_p, x = x_p$; and (3) along the line to find v_p, as following:

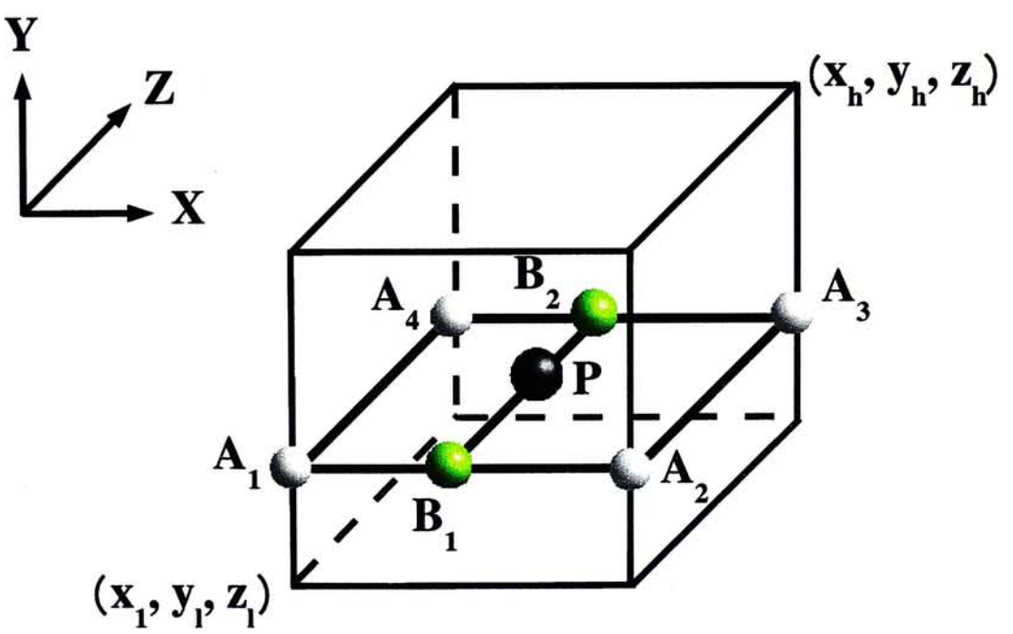


Figure 1.1: Trilinear interpolation.

$$
\begin{aligned}
v_{A_1} &= (y_p - y_l) * S(x_l, y_h, z_l) &+ (y_h - y_p) * S(x_1, y_l, z_l) \\
v_{A_2} &= (y_p - y_l) * S(x_h, y_h, z_l) &+ (y_h - y_p) * S(x_h, y_l, z_l) \\
v_{A_3} &= (y_p - y_l) * S(x_h, y_h, z_h) &+ (y_h - y_p) * S(x_h, y_l, z_h) \\
v_{A_4} &= (y_p - y_l) * S(x_l, y_h, z_h) &+ (y_h - y_p) * S(x_1, y_l, z_h) \\
v_{B_1} &= (x_p - x_l) * v_{A_2} &+ (x_h - x_p) * v_{A_1} \\
v_{B_2} &= (x_p - x_l) * v_{A_3} &+ (x_h - x_p) * v_{A_4} \\
v_p &= (z_p - z_l) * v_{B_2} &+ (z_h - z_p) * v_{B_1}
\end{aligned}
\tag{1.1}
$$

Besides the date value v, it is often useful to find gradient g of a point at the volumetric data. The gradient can be used to estimate the surface normal direction since the gradient is perpendicular to surfaces of constant value. Moreover, the gradient can also be used to describe the opacity for volume visualization or the stiffness for haptic simulation. The gradient $g = (g_x, g_y, g_z)$ at (x, y, z) is generally estimated by taking central differences:

$$
\begin{aligned}
g_x &= \frac{1}{2}(f(x+1, y, z) - f(x-1, y, z)) \\
g_y &= \frac{1}{2}(f(x, y+1, z) - f(x, y-1, z)) \\
g_z &= \frac{1}{2}(f(x, y, z+1) - f(x, y, z-1))
\end{aligned}
\tag{1.2}
$$

1.2 Volume Visualization

Over the years, many techniques have been developed to visualize volumetric data. To display regions of constant value within the volumetric data, indirect rendering techniques generate geometric surface models from the data. To display the whole volumetric data, direct rendering techniques were developed that attempt to capture the entire data in a single 2D image.

Depending on the type of interpolation function used, there are two approaches to represent regions of constant value within the volumetric data by geometric model. If a zero-order interpolation function is being used, the generated structure is a set of voxels with the interested value. This approach is known as the *cuberille* methods [47]. If a higher interpolation functions is being used, an *isovalued surface* or *isosurface* pass through the samples is generated. This approach is known as *isosurface extraction* methods [24, 5, 33]. The main disadvantage of the cuberille methods is that the image produced is generally blocky, because the structure generated is a set of orthogonal

faces. Isosurface extraction methods generate surface patches with any orientation, so that they have the potential of providing a closer approximation to the underlying surface. However, this greater generality may carry with this approach greater costs in both processing times and storage space. Moreover, since it attempts to generate a surface in more detail than the data samples, it has the ambiguity of isosurface structure in some cases. To deal with the problems, many algorithms are proposed to simplify the geometric model [6, 13, 42, 49], or to address the problem of ambiguity [10, 30].

Volume rendering [17] is the process of creating a 2D image directly from 3D volumetric data. It is useful to display information contained in the volumetric data that cannot be adequately represented using surface, or information is contained not only on the surface. Volume rendering can be achieved using an *object-order*, an *image-order*, or *domain-based* technique. Object-order volume rendering techniques use a forward mapping scheme where the volumetric data is mapped onto the image plane. In image-order algorithms, a backward mapping scheme is used where rays are cast from each pixel in the image plane through the volumetric data to determine the final pixel value. In a domain-based technique the spatial volumetric data is first transformed into an alternative domain, such as compression frequency, and wavelet, and then a project is generated directly from the domain. Although volume rendering methods can convey more information inside volumetric data, the huge size of volumetric data (for example, 16 Mb of memory is required to store a medium resolution data of 256^3 samples of one byte precision), increase algorithm complexity, and consequently increase the rendering time. To improve interaction in volume rendering, many optimization methods as well as several special-purpose volume rendering hardwares have been developed [18, 48, 32, 31].

1.3 Thesis Contributions

The main contribution of this thesis are techniques to convey information in volumetric data, in particular the isosurface structure, both in visual and haptic terms.

The first contribution is a new technique, named *multi-body surface extraction* for generating a multi-body surface from volumetric data in a single processing step. For some applications, in anatomy, geology, etc., it is not enough to find the boundary of a single object from the volumetric data. One needs to extract a system of objects in contact, for example, the boundaries of muscle and bone from the rest of a CT image. Applying traditional isosurface methods several times to extract those surfaces and

then editing them to match each other, is a laborious and painful task. Moreover, there are many situations in which a precise fitness between objects is desirable. Hand editing causes technical difficulties and may create defects in the pictures generated. Moreover, even when the fitness between two objects is perfect, it is a waste of graphical resources to draw the boundary between them twice, and makes rendering slower. Our algorithm builds the isosurfaces upward dimensionally, which extends naturally to the multi-body structure. It can efficiently create a usable multi-body structure by a single pass through the volumetric data.

The second contribution is haptic rendering techniques of volumetric data. With the development of volume visualization methods, we can easily display meaningful information from volumetric data using interactive graphics and imaging. Haptic interaction with volumetric data adds a new modality to volume visualization, which has an advantage in presenting complex attributes of local region. Haptic rendering refers to the computational methods used to determine the forces that resulted when we interact with virtual object [1, 25, 36, 50]. Although the benefits of haptic simulation of volumetric data have been recognized recently [16, 2, 14, 3], this area of research has not yet been fully explored. Most traditional haptic rendering methods are developed to compute realistic interaction force with geometric primitives. Direct volume haptic rendering allow haptic palpation of volumetric data, but lack of the ability of simulating the contact sensation of stiff embedded implicit surface. We proposed a direct haptic rendering method of isosurface in volumetric data using a point-based haptic feedback device, without the extraction of the isosurface to geometric representations such as polygons. Our algorithm uses a virtual plane as an intermediate representation of the isosurface, and computes the point interaction force based on this virtual plane.

In conclusion, using the proposed methods, we are able to build multi-sensory virtual environment for interaction with volumetric data.

1.4 Thesis Outline

This thesis is divided into two main parts. **Part I** describes our brand new multi-body surface extraction algorithm for volume visualization and **Part II** describes methods for haptic rendering of volumetric data. Readers are referred to **Chapter 2** and **Chapter 4** for a quick overview of the motivations of our work.

Part I begins with an introduction to the isosurface extraction techniques for volume visualization and an overview of previous work in **Chapter 2**. We will also

discuss the motivation of our research to introduce a multi-body surface extraction algorithm. The details of the new algorithm will be discussed in **Chapter 3**.

Part II begins with an introduction to the *Haptics* – the study of haptic interaction of human with objects, in **Chapter 4**. We will clarify the terminology concerning both human and machine aspects in this field, and also discuss the haptic rendering process of graphic models. Our motivation is to develop methods for haptic rendering techniques of volumetric data. Since most traditional haptic rendering methods are designed for geometric models, the main approaches will be discussed in **Chapter 5**. The haptic rendering methods of volumetric data will be discussed in **Chapter 6**. Our proposed algorithm is a direct haptic rendering method of isosurface in the volumetric data using an intermediate representation of virtual plane.

Finally, we conclude the work of this thesis and give future directions in **Chapter 7**.

Part I

Multi-body Surface Extraction

Chapter 2

Isosurface Extraction

Methods for displaying polygons are well developed to generate synthesized images in computer graphics. There are software techniques for accelerating the rendering speed (such as hidden surface removal, Z-buffer, polygon scan conversion, etc.), or for improving the realist of the images (such as texture mapping, etc.) [7]. There are also some specialized hardware developed such as Silicon Graphics (SGI™) workstations. Even on PC, there are add-on cards for accelerating surface rendering with OpenGL® library support. Therefore, regions of constant value, which represents an implicit surface structure within a volumetric data is generally approximated using geometric primitives. The surface structure is defined by first applying a binary segmentation function $B(v)$ to the volumetric data. $B(v)$ evaluates to 1 if the value v is considered part of the object, and evaluates to 0 if the value v is part of the background. The surface is then the region where $B(v)$ changes from 0 to 1.

$$B(v) = \begin{cases} 1, & \text{if } v \geq \tau, \text{ where } \tau \text{ is the value of the surface.} \\ 0, & \text{otherwise.} \end{cases} \tag{2.1}$$

If zero-order interpolation functions are being used, the *cuberille* methods represent the constant value regions by a set of cubes that include the values of interests. An early algorithm [47] displays human organs from CT data using a set of shaded cubes. Each cube is rendered by standard surface rendering algorithm and hidden surface removal is done by the Z-buffer algorithm. The main disadvantage of the algorithm is that the image produced is generally blocky, which can be improved by surface shading algorithm or low-pass filtering.

If continuous interpolating functions are being used, *isosurface extraction* methods generate surface patches with any orientation, which can provide a closer approximation to the underlying surface. Early isosurface extraction approaches first generated

two-dimensional contour lines for parallel slices running through the data, and then connected the contour lines into three-dimensional isosurfaces [9]. There are two main steps in such algorithms: first, closed contours from each slice are extracted by some edge detection algorithms; second, a contour joining strategy is used to find the optimal triangular strips to fit adjacent contours. The main drawback of this approach is that the joining of contours is complex. One slice may contain two or more contours of the desired surface. A branch would occur if two adjacent slices contain different number of contours.

More recent approaches create the isosurface by examining the three dimensions at once. In these approaches, the generation of an isosurface involves determining, for each cell, whether the isosurface crosses the cell, and if so, approximate where the isosurface lies. By using the binary segmentation function $B(v)$, the eight vertices of a cell is classified as inside the isosurface if $B(v) = 1$, or outside the isosurface if $B(v) = 0$. For the brevity of the discussion below, we denote these $\bullet$ and $\circ$. When some vertices of a given cell are $\bullet$ and some $\circ$, the isosurface must pass through the cell, and the problem becomes one of finding where it does so. An intersection point is the point at which the isosurface is estimated to cross the edge connecting two adjacent cell vertices those have different labels, and whose positions are approximated by linear interpolation. Such intersection points will become vertices of one or more polygon of the isosurface whose edges lie in the faces of the cell. Finally, the extracted isosurface (denoted by Σ) is composed by all the polygons generated within each cell.

In next section, **Section 2.1**, we will discuss some well-known algorithms for isosurface extraction. We state the motivation of our research in **Section 2.2**.

2.1 Previous Works

2.1.1 Marching Cubes

Marching Cubes [24] (MC) is the widely implemented isosurface extraction method, which uses a set of triangles to represent the isosurface. The algorithm determines how the surface intersects with a cell (*cube*), then moves (*matches*) to the next cube. Since there are 8 vertices in each cube and two states, $\bullet$ or $\circ$, there are only $2^8 = 256$ ways a surface can intersect the cubes. Furthermore, two different symmetries of the cube reduce the problem from 256 cases to 15 patterns: first by interchanging the $\bullet$ and $\circ$, and second by rotational symmetry. **Fig. 2.1** shows these 15 patterns, where polygons of more than three edges are normally non-planar, and often tessellated into

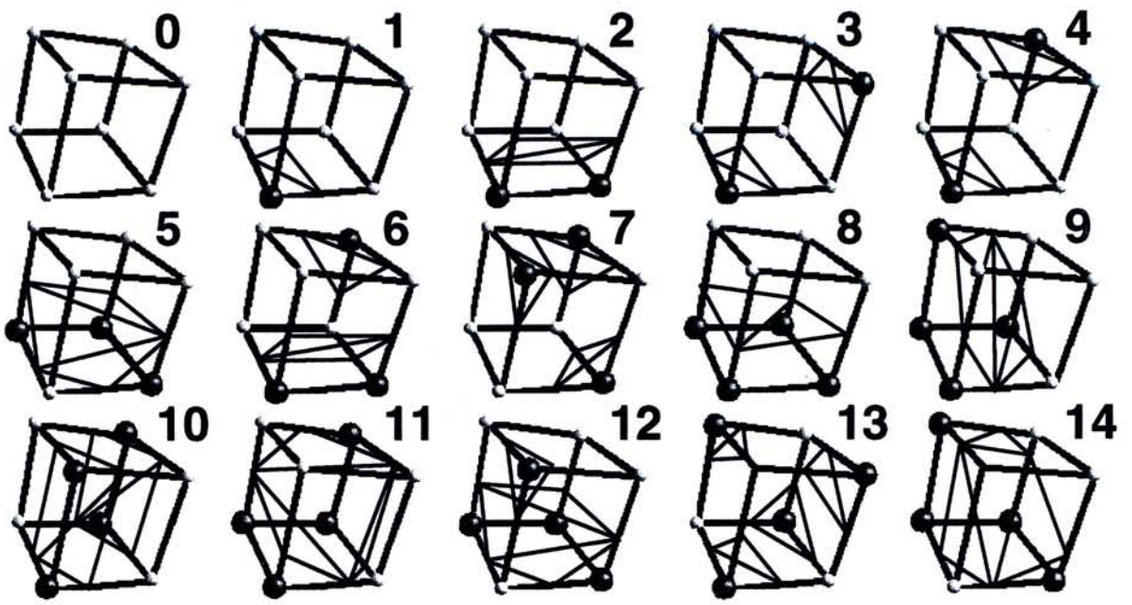


Figure 2.1: The 15 patterns of Marching Cubes (MC) algorithm.

triangles. An index is created by using the states of the vertices. This index serves as a pointer into a look-up table, which gives all edges the isosurface intersecting with the cube and the triangulation, with the actual triangles vertices approximated using linear interpolation. The final step of the algorithm is to estimate a normal vector for each triangle vertex using central differences. The output triangles with vertex normals can be rendered by standard graphic hardwares to get a smooth shaded image of the isosurface.

There are two main problems of the MC algorithm. First, it often generates a huge number of small triangles, which require significant amount of time for rendering. Therefore, simplification algorithms are proposed which are either post-processing methods that alter or combine the resultant polygons in order to reduce the number of polygons and smooth the isosurface [6, 13, 42, 49], or methods that directly generate smaller number of polygons by taking advantages of viewing information [22] or the underlying isosurface structure [44, 43, 34]. Second, the severer problem is the isosurface generated may contain holes or noise due to ambiguous cases during the extraction process and such errors are intolerable in some applications such as medical diagnosis. **Fig. 2.2** shows the ambiguity problem. There are different ways of triangulation of some cases in **Fig. 2.1** (3, 6, 7, 10, 12, 13), and inconsistent choice can result in holes in the generated isosurface. The problem is due to the fact that MC algorithm attempts to generate a surface in more detail than the data samples, and

is theoretically insolvable. Some papers study the problems of disambiguation, report on some heuristics, and present some statistics on the occurrence of such ambiguities [10, 30].

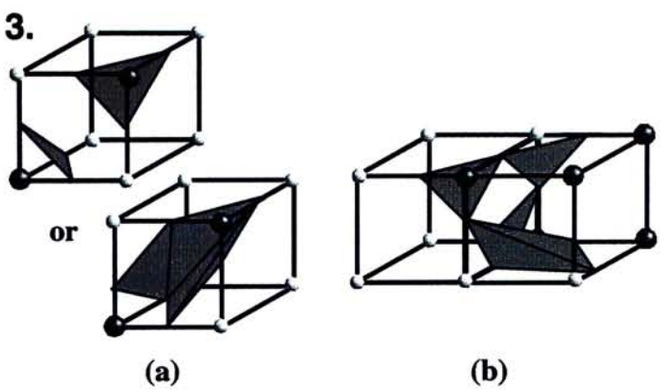


Figure 2.2: The ambiguity problem of MC algorithm: (a) There are different ways of triangulation for some cases, (b) Inconsistent choice can result in holes.

2.1.2 Skeleton Climbing

Instead of generating the isosurface from volumetric data by building triangulated surface directly within each cube formed by eight samples, *Skeleton Climbing* [33] (SC), constructs the isosurface upward dimensionally, by first finding iso-points on cube edges (elements of 0-skeleton of Σ), then iso-lines on faces (elements of 1-skeleton of Σ) and finally isosurfaces within cubes (elements of 2-skeleton of Σ), as shown in **Fig. 2.3**. Although it builds the isosurface in different ways with MC, SC has the advantages of MC that it is parallelization ready. Both algorithms use table lookup technique, therefore, SC runs at a speed little different from MC. Besides, SC algorithm also allows triangle merging at some edges, and thus has a 25% reduction in number of triangles generated [33].

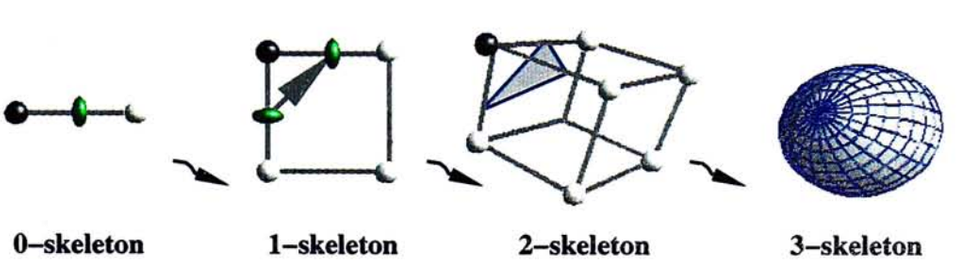


Figure 2.3: Overview of the Skeleton Climbing algorithm.

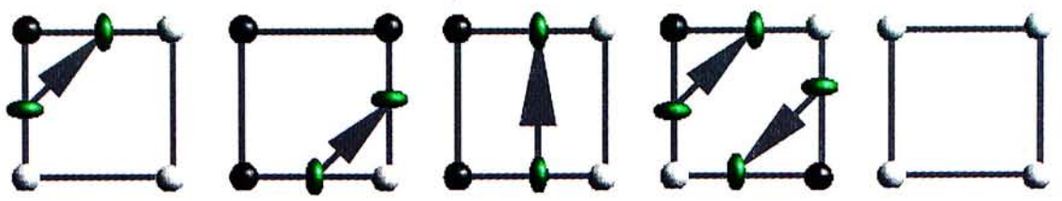


Figure 2.4: Five cases of two-type faces.

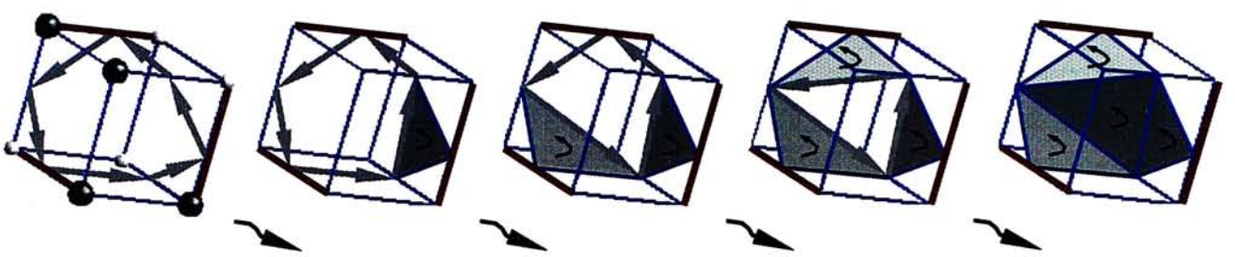


Figure 2.5: The 'nibbling corner' process for triangulation.

Isosurface vertices are determined just as MC does, one on each cube edge that joining a ∘ to a •. Those edges are called *occupied edges*, and must contain vertices whose positions are computed by linear interpolation. Rather than go directly to cubes as MC does, SC then constructs the surface edges joining vertices, in a cube face. The five possible cases of cube faces are shown in **Fig. 2.4**. Edges are drawn as arrows. Following the direction of the arrow, the •s are bound to the left. The four vertices on the edges of a $\begin{smallmatrix}\circ & \bullet\\ \bullet & \circ\end{smallmatrix}$ face can joined in two ways. The choice can use interpolation schemes, but all current methods assume that the sampled function is locally polynomial [10, 30] in a cube. However, when the actual surface is discontinuous, such as that between the density of bone and the density of soft tissue, the results are not especially meaningful. As long as the surface as a whole is smooth on the cube scale, the choice is typically without topological effect. The simple rule, "do not cross ∘–∘ diagonals", is as sensible as any more complex choice, and efficient for speed. The MC algorithm directly step to cube will reduce the number of cases both by rotational symmetry and by interchanging ∘ and •. However, this may mismatch edges in a sharing $\begin{smallmatrix}\circ & \bullet\\ \bullet & \circ\end{smallmatrix}$ face between adjacent cubes, giving holes in the surface. Building the 1-skeleton first, and reading it from both cubes sharing the face avoid the problem.

The next step is to find triangles (elements of the 2-skeleton of Σ) in each cube of volumetric data. After building the edges in the faces, there are loops formed on cubes. The triangles are constructed by 'nibbling corners' of the edge loops, as shown in **Fig. 2.5**. If a n-sided polygon is filled by triangles without introducing new vertices,

there are $n-2$ triangles. For example, if the loop has six sides, there are four triangles required to fill it. Each step shrinks the set of vertices by one, and three vertices leave a single triangle. There is a large set of possible edge choice sequences for applying the nibbling process and hence a large class of algorithmically distinct ways to assign a minimal triangle set, though not always with distinct result. Any sequence gives the same number of triangles within the cube, as MC does. However, different orders will give different geometry of the final isosurface. In the worst case, where the triangles are emitted using a *fan* strategy, with all interior edges radiating from the first vertex. This produces ugly forms, with long thin triangles, resulting in a greater edge length and a more wrinkled surface than necessary. A better triangular mesh closely approximates the true isosurface. This can be done by minimizing the deviation between the planar normal of the triangle and the gradients at its three vertices computed by central differences.

2.1.3 Adaptive Skeleton Climbing

Adaptive Skeleton Climbing [34] (ASC), using the isosurface construction method of SC, but generates triangles from boxes whose size will adapt to the geometry of the isosurface. ASC is able to extract a smooth isosurface with between 4 and 25 times fewer triangles than marching cubes algorithm in comparable running times.

The problem of MC is that it subdivides the volumetric data into small unit sized cubes and then generates triangles within the cubes. Thus, even the enclosed isosurface is smooth enough to be approximated by larger triangle; many small triangles are still generated. Although mesh simplification algorithms [6, 13, 42, 49] can reduced the triangle counts, a better idea is to generate isosurface adaptively with on-the-fly triangle reduction. It is generally faster and more accurate because it directly makes use of the volumetric data. Adaptive marching cubes approaches [44, 43] have been proposed which used octree to partition the volumetric data. The idea is to fit smooth region with large cubes while complex region with small cubes. However, octree is still quite restrictive in partitioning the volume, since the partitioned region should always be a cube. Moreover, the resultant isosurface may contain gaps that exist between a large box and its small neighbors. ASC uses binary tree organization along each dimension of the partitioned boxes, and thus allows more flexibility in partitioning the volume. It prevent the formation gaps of the isosurface by sharing information among neighbor boxes.

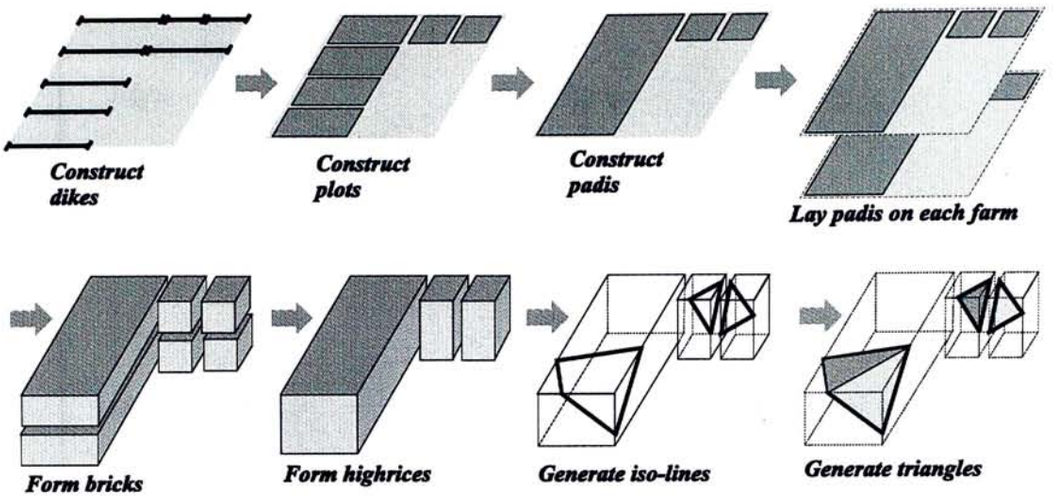


Figure 2.6: Overview of Adaptive Skeleton Climbing (ASC).

Volume Partitioning

Instead of partitioning the volumetric data in a top-down approach, ASC builds size-maximal box that enclose the smooth isosurface bottom-up. The goal is to fit smooth region with large boxes, by analysis of the volumetric data. **Fig. 2.6** shows the process of adaptive skeleton climbing graphically, and **Fig. 2.7** shows the glossary of the data structure.

The volumetric data analysis starts in 1D by considering a linear sequence of samples. A line of 2^n+1 data samples is called *lign*, where n is an integer ≥ 0. A *dike* is a segment of lign which covers data samples in the interval $[a2^m, (a+1)2^m]$, where $0 \leq m \leq n$ and $0 \leq a < 2^{n-m}$, both a and m are integers. That is, all dikes are organized in a binary tree. A *simple dike* is crossed at most once by the isosurface. The goal is to find length-maximal simple dikes inside a lign.

In 2D, considering a $(N+1) \times (N+1)$ *farm* of samples, with $N+1$ horizontal and $N+1$ vertical ligns, a *strip* consists of two consecutive ligns and a *plot* is analogous to dike which consists of two consecutive dikes. A rectangle with dikes as sides is called *padi*. The goal is to subdivide the 2D farm into size-maximal *simple padis*. First, *simple plots* are found. A plot is simple if and only if its two dikes are also simple. Then neighboring simple plots are merged to form simple padis, as large as possible. A padi is simple if all plots inside it are simple and its four side dikes are simple. Once the size-maximal padis within a farm is found, iso-lines can be generated as SC.

With these 1D and 2D data structures, enough information is provided to construct 3D simple boxes. Consider a *block* of $(N+1) \times (N+1) \times (N+1)$ samples, a *slab*

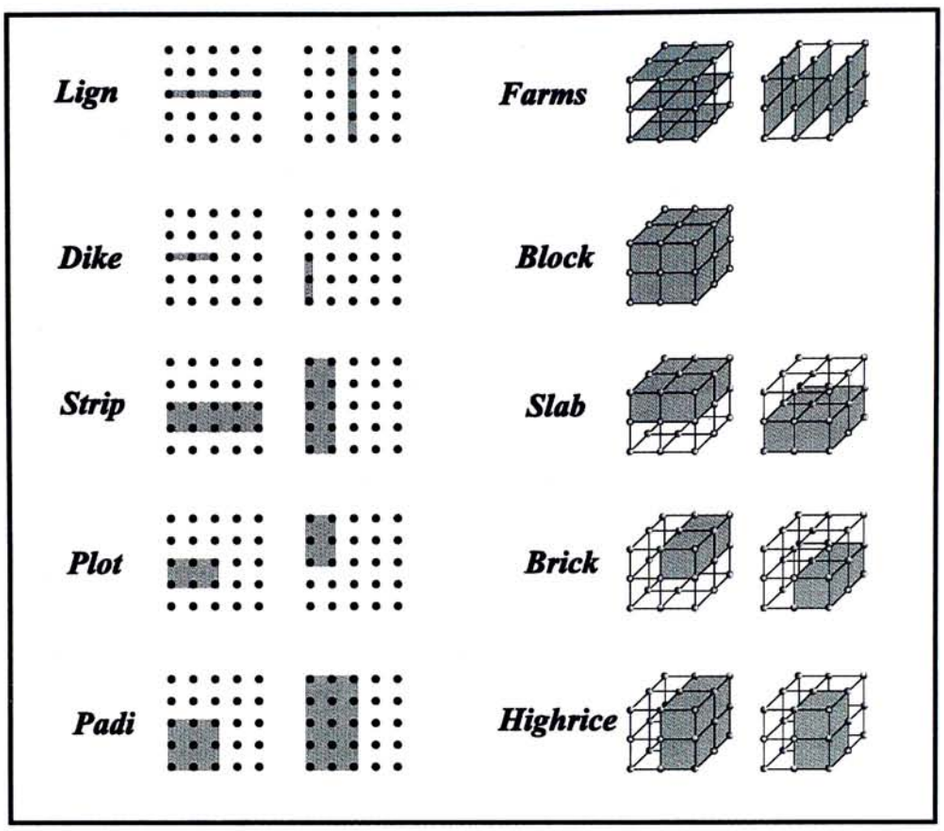


Figure 2.7: Glossary of various ASC data structure.

being analogous to a strip containing two consecutive ligns, consists of two consecutive farms. A *brick* in the slab has two matching padis in two consecutive farms as faces. A *highrice* is a rectangular box composed of stacked bricks. The goal is to find out size-maximal *simple highrices* in the block. First, *simple bricks* are found. A brick is simple if the two padis forming it are also simple. Simple bricks are stacked one by one to construct the simple highrice in a particular direction. A highrice is simple if all its component bricks are simple and its six faces are simple padis. Once the volumetric data has been partitioned into size-maximal highrice, triangular mesh can be generated within each highrice. Since the size of the highrice reflects the geometry complexity of the enclosed isosurface, hence the generated triangles will also adapt to the geometry. That is, large triangles are generated to approximate smooth isosurface regions.

Triangular Meshes Generation

If the triangular mesh is generated immediately inside each highrice, cracks will be resulted. **Fig. 2.8(a)** shows a large highrice next to a small one. The isosurface crosses the plane separating the two highrices, as shown in (**Fig. 2.8(b)**). If triangles

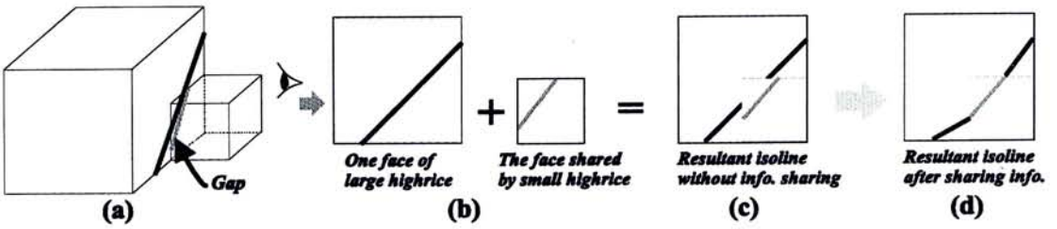


Figure 2.8: Sharing information between neighbor highrices.

are emitted for each highrice without the knowledge of their neighbors, gaps will appear in the generated triangular mesh. This is because the linear iso-lines generated on the highrice surfaces may not match each other geometrically, as shown in **Fig. 2.8(c)**). To prevent this mismatch, information must be shared between adjacent highrices, as shown in **Fig. 2.8(d)**).

After the edge loops are generated on surfaces of each highrice, triangles can be emitted as the 'nibbling corners' processing of SC. Triangulation of the edge loops is done by minimizing the deviation between the planar normal of a triangle and the gradients at its three vertices, in order to obtain triangular mesh which closely approximate the true isosurface.

2.2 Motivation

We noticed that there is another shortage of the traditional isosurface extraction algorithms. Existing algorithms often employ a binary segmentation to the volumetric data, and thus only extract a single isosurface. For volumetric data that contain multi-object structure, as in anatomy, geology, etc., most of the information is lost. One needs to extract a system of objects in contact, for example, the boundaries of muscle and bone from the rest of a CT image. We can apply traditional isosurface methods several times to extract those surfaces and then edit them to match each other manually. However, it is a time demanding processing, as we are required to process the huge-sized volumetric data several times. Moreover, hand editing is a laborious and painful task, and inevitably causes technical difficulties if precise fitness between objects is desired. Furthermore, even when the fitness between the objects is perfect, it is a waste of graphical resources to draw the boundaries between them twice, and makes rendering slower.

To extract multi-object structure, we apply n-type segmentation instead of binary segmentation to the volumetric data. It would be very hard to adapt MC to create a

multi-object structure. MC is based on a list of cube cases, which becomes far more complex if multiple bodies are allowed, since the n^8 possibilities grow rapidly from the manageable $2^8 = 256$ to $3^8 = 6561$, $4^8 = 65,536$, and so on. SC's approaching of building isosurface upward dimensionally, with surface construction on the fly, makes a natural setting for the task.

Our algorithm is a generalization of the SC algorithm, which creates efficiently a usable multi-body structure by a single pass through the volumetric data. The volumetric data is handled block by block with the size of $(N+1) \times (N+1) \times (N+1)$. The blocks are classified as binary (at most two types of samples) or non-binary. For the binary block, ASC is applied to extract the surface structure effectively by adapting to the underlying isosurface. For the non-binary block, we build the isosurface using SC's approach, but more cases will be involved as more than two types of samples are allowed to meet in a face or a cube. We will discuss the details of multi-body surface extraction in next chapter.

Chapter 3

Multi-body Surface Extraction

The Multi-body Surface Extraction algorithm is a generalization of the Skeleton Climbing algorithm [33, 34], which extracts a multi-body surface from a volumetric data in a single-pass of processing. This chapter describes the details of the isosurface generation process for cubes that have more than two types of samples occur at its corners.

This chapter is organized as follow. We will first introduce the multi-body surface extraction in **Section 3.1**, including the problem, the assumptions and the basic building elements. Then we will consider the building of 0-skeleton (vertices separating different types of samples) in **Section 3.2**. In **Section 3.3**, we will discuss the building of 1-skeleton (iso-lines on or inside the cubes joining vertices). A general scheme for building 1-skeleton elements inside messy cubes (cubes with not less than four types of samples occur at its corner) will be described in **Section 3.4**. Afterwards, we will present the triangular mesh generation – the building of 2-skeleton (isosurfaces within each cube) and the building of 3-skeleton (the resultant multi-body surface), in **Section 3.5**. Finally, we present the implementation and results in **Section 3.6**.

3.1 Multi-body Surface

Suppose we are given a volumetric data with several interested objects defined by non-overlapping ranges of values, a multi-body surface separates them from each other. If the samples can be classified into types $0, ..., N$, there are at most $\frac{N(N-1)}{2}$ isosurfaces. A particular isosurface Σ_{ij} separates type i from type j, where $i < j$. There is a normal at every point of Σ_{ij} (in geometric approximation, at each vertex), and type i is bounded in the positive direction of the normal and type j in the other side. The

multi-body surface is the union of those isosurfaces:

$$\Sigma = \bigcup_{i=0}^{N} \bigcup_{j=i+1}^{N} \Sigma_{ij} \tag{3.1}$$

Traditional isosurface extraction algorithms (such as [24, 5, 33]) can only extract a single isosurface from a volumetric data at each time. Therefore, to extract the multi-body surface, we must apply the algorithm several times in order to obtain all the isosurfaces of N types and then edit them to match each other manually. However this approach suffers from several disadvantages. First, a lot of time is required to process the huge-sized volumetric data several times. Second, the hand editing process is laborious, painful, and may cause technical difficulties if precisely fit boundaries between objects are desired. Lastly, even when the boundaries between objects are perfectly fit, it is a waste of graphical resources to draw the boundaries between them twice, and slows down the rendering process. Therefore, a better approach is required that the multi-body surface can be extracted directly from the volumetric data by one single processing step, and also to create precisely fit boundaries between the isosurfaces.

As the multi-body surface is far more complex than a single surface, some reasonable assumptions must be made in order to extract the isosurfaces efficiently. We will first assume that the surface between two types is generally smooth, without corners or sharp ridges. This is generally true for most volumetric data. However, for some cases, we do have to get a surface with fractures, such as the boundary between air and tooth in a CT data. However, we can still let it to be a post-processing step. It is because, recognizing those areas are hard at cube level. They are more easily seen once a boundary surface is created, and can be found at regions where the curvature are high.

Besides smoothness, we also assume that a particular isosurface extracted is an manifold, and multiple isosurfaces meet in stable, generic ways. Therefore, in 2D, two objects meet in a smooth non-self-crossing curve. At a region where three objects meet, there must be at least one point in contact with them. The point is known as *tripoint*, as shown in **Fig. 3.1(a)**. Similarly, there must be one point in contact where four objects meet. However, we can always replace a region where four objects meet, with two regions where only three objects meet without changing the values sampled at the four corners of a cube face. Any quadruple point can thus be replaced with two tripoints, as shown in **Fig. 3.1(b)**. Therefore, unless there are strong reasons to assume that a special case holds, it is reasonable to build multi-body 2D boundaries using only smooth curves and tripoints. Analogously, in 3D we can approximate any multi-

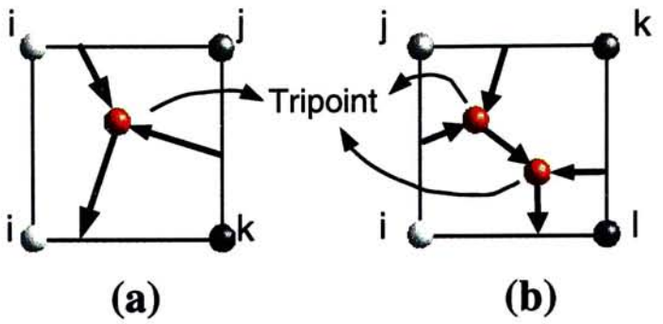


Figure 3.1: Multi-body 2D boundary building element: (a) A tripoint separates three types, (b) Two tripoints separate four types.

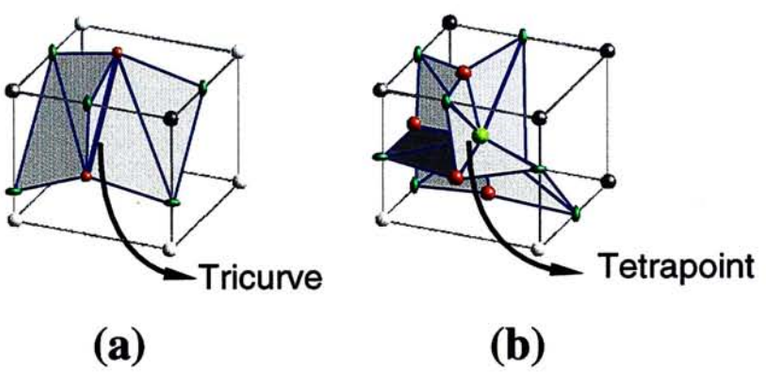


Figure 3.2: Multi-body surface building elements: (a) Tricurve, (b) Tetrapoint.

body boundary, to within cube precision, using smooth surfaces that meet in *tricurves* (iso-lines that joining two tripoints/tetrapoints) and *tetrapoints* (in-cube vertices that connect to tripoints or other tetrapoints), as shown in **Fig. 3.2**. They are the basic building elements for a multi-body surface and everything else can be approximated by something built with them.

3.2 Building 0-skeleton

Assume the volumetric data is *cartesian*, with the sample values on an integral grid – $[0, L] \times [0, M] \times [0, N]$. At each sample location (l, m, n), we have a value $v = f(l, m, n)$. In the simplest case this will be a scalar, such as the density reported by a CT or MRI scan. However, MR spectroscopy, or light photography of physically sliced data, can produce multi-component samples, such as, $f(l, m, n) = ((f_R(l, m, n), f_G(l, m, n), f_B(l, m, n))$ color values.

If the volumetric data contains more than one object of interest, for simplicity, considering in the scalar case, we will often have a set of thresholds to classify the samples into N types, such that:

$$\begin{aligned} &\tau_0 = -\infty \leq \tau_1 \leq, \ldots, \leq \tau_{N-1} \leq \tau_N = \infty, \\ &(l,m,n) \text{ is of type } i \Longleftrightarrow \tau_{i-1} < f(l,m,n) < \tau_i \end{aligned} \tag{3.2}$$

This classification is easily extended to multi-component samples.

The first step of the isosurface extraction is to find the 0-skeleton. An *occupied edge* joins two neighboring samples (say, at (l,m,n) and $(l+1,m,n)$), of different types. The actual position of the separating vertex at the occupied edge can be computed by linear interpolation. In the case where the two samples are classified as type i and $i+1$, we can compute the separating vertex just as in the single body case, at the point:

$$\left(l + \frac{\tau - F(l,m,n)}{F(l+1,m,n) - F(l,m,n)}, m, n\right), \ with\ \tau = \tau_i \tag{3.3}$$

Vertices at the occupied edges in the m and n directions are computed similarly.

Where the samples are classified as type i and $j > i+1$, any value in the range from τ_i to τ_{j-1} is a candidate to replace the threshold τ in **Eqn. 3.3**. However, the choice cannot be made on abstract grounds, in a general way, if precise boundaries between the types are desired. For example, if type j represents bone and should have a smooth surface, we should use $\tau = \tau_{j-1}$ independently of what other type is being meet. However, if type i is preferred for smoothness, we use τ_i independently of j. Therefore, we set a default value $\tau_{ij} = (\tau_i + \tau_{j-1})/2$ for the threshold to use where types i and j meet, but also provide an user interface to modify these values according to domain expertise. By experiment, it shows that the visual differences of the rendered images of the surfaces extracted by using different thresholds, namely $(\tau_i + \tau_{j-1})/2$, τ_{j-1}, τ_i, are not significant. It is however, when precise boundaries of different types are desired, it is still necessary to provide an user interface to modify the thresholds. To distinguish the vertices of different isosurfaces, each computed vertex is associated with a label (i,j), where $i < j$, if it separates type i from type j.

Unlike single-body surface, 0-skeleton of multi-body surface includes also *tripoints* and *tetrapoints*, located at where more than two types of objects meet. They do not occur at cube edges. Tripoints occur at cube faces, and tetrapoints occur inside cube volume. We will discuss the computation of the positions of the tripoints and tetrapoints at next section, where we will show that the introduction of them is necessary for building the 1-skeleton. Each tripoint is associated with a label (i,j,k), where

$i < j < k$, if it separates types i, j and k. Similarly, each tetrapoint is associated with a label (i, j, k, l), where $i < j < k < l$, if it separates types i, j, k and l.

The elements of the 0-skeleton will be the vertices of the final triangular mesh. Having located the vertices, the next step is to compute the normal vectors at each vertex, so that the triangular mesh can be rendered correctly by surface shading algorithms. At a vertex where only two types meet, the unique normal can be approximated by central differences. However central differences cannot be used to approximate a single normal vector at a tripoint or a tetrapoint. It is because, more than two types meet at those points, and therefore, more than one normal vector should be associated with them. A tripoint touching types i, j and k has three normals $\mathbf{N}_{ij}, \mathbf{N}_{jk}, \mathbf{N}_{ik}$, pointing from type j to type i, type k to type j, and type k to type i respectively, where $i < j < k$. We need these distinct normal vectors when rendering the surfaces with Gouraud or Phong shading. The use of any one normal for all three surfaces touching the same tripoint will give an effect of high curvature on at least two of them. Moreover, a shaded triangle may look strange, when the normals at its vertices are far apart in direction. Similarly, a tetrapoint touching four types $i < j < k < l$, has six normals $\mathbf{N}_{ij}, \mathbf{N}_{ik}, \mathbf{N}_{il}, \mathbf{N}_{jk}, \mathbf{N}_{jl}, \mathbf{N}_{kl}$. Each of the three (respectively six) normals can be constructed by using the vertex normals of the triangles touching the vertices. We will discuss it in more detail at **Section 3.5**.

3.3 Building 1-skeleton

After building the 0-skeleton of the multi-body surface, the next step is the building of the 1-skeleton. The 1-skeleton is a set of oriented edges joining the elements of 0-skeleton. The algorithm extends the SC algorithm, but with multiple objects, one can expect more cases should be considered.

We called a cube binary if at most two types i and j occur at its eight corners. Evidently for a binary cube, we can create oriented edges just as SC does (where type $i = \circ$, and type $j = \bullet$). In non-binary cubes, there are different types of vertices with different labels. Therefore, we should not connect the vertices arbitrarily to form edges, but according to their types and labels:

- When we join vertices of same type, such as a normal vertex with a normal vertex or a tripoint with a tripoint, we only join vertices of same labels. An exception is that there are edges joining tetrapoints of different labels. However, we will show that an edge joining two tetrapoints is actually a result of the merging of

two edges joining a tripoint to those tetrapoints and the deletion of that tripoint. Therefore, we do not actually create an edge joining two tetrapoints.

- When we join a normal vertex with a tripoint, the label of the normal vertex should be a subset of the label of the tripoint.
- When we join a tripoint with a tetrapoint, the label of the tripoint should be a subset of the label of the tetrapoint.
- We will not join a normal vertex to a tetrapoint to form an edge, because tetrapoints are introduced inside cubes that only connect to tripoints or other tetrapoints.

The edges created are oriented so that when seen from the outside of a cube, the higher type is bounded at the left of the edge. There are two types of edges. First, a normal edge connects a normal vertex to either a normal vertex or a tripoint. Second, a *tri-edge* connects a tripoint to either a tripoint or a tetrapoint. Moreover, the edges are labeled to distinguish that they are separating different types of objects:

- The label of an edge joining two vertices of same type has the label of the vertices. Except that the label of an edge joining two tetrapoints has the label of the deleted tripoint.
- The label of an edge joining a normal vertex with a tripoint is the label of the normal vertices.
- The label of an edge joining a tripoint with a tetrapoint has the label of the tripoint.

At the following sub-sections, we first discuss the building of the normal edges at the cube faces, then discuss the building of tri-edges inside the volume of complex cubes.

3.3.1 Non-binary Faces

Many faces of a non-binary $2 \times 2 \times 2$ cube will be binary, and thus can be treated in the same way as SC algorithm, producing vertices on cube edges and 1-skeleton elements to join them. There are five cases of binary faces, and the edges are connected as SC does, as shown in **Fig. 2.4**. If there are more than two types occur at the four corners of the cube face, the face is non-binary. There are two types of non-binary faces, depending on how many types occur at face corners.

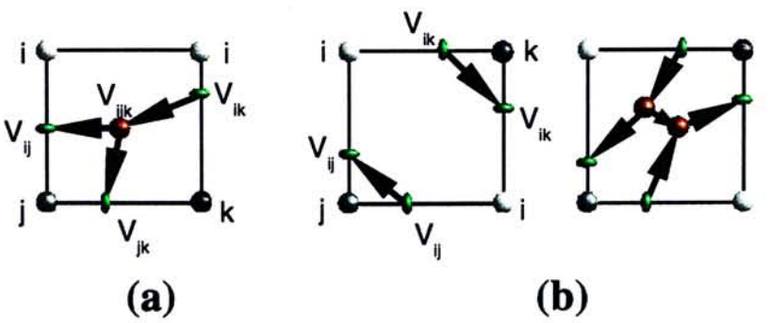


Figure 3.3: Three-type face topology: (a) Two type-i corners are neighbors, (b) Diagonally opposite.

Three-type faces

If three types i, j, k are distributed among four face corners, two corners must be the same type, for example, both type i. If these two corners are neighbors, the corresponding cube edge has no 0-skeleton elements. We find the separating vertices at the other three occupied edges by linear interpolation: $\mathbf{V}_{ij}$ at the edge joining type i with j, $\mathbf{V}_{jk}$ at the edge joining type j with k, and $\mathbf{V}_{ik}$ at the edge joining type i with k. We are now facing the planar problem of how three types are separated on the cube face. We know that, at the region where three objects meet, there must be at least one point in contact of the objects. We named the point as *tripoint*, and it must lie at somewhere inside the cube face, connecting to $\mathbf{V}_{ij}, \mathbf{V}_{ik}$ and $\mathbf{V}_{jk}$, as shown in **Fig. 3.3(a)**. We also know that, in order to obtain a smooth isosurface, the triangles generated should have a well-formed shape without long edges and sharp corners. Therefore, we assign the (x, y, z) position to the tripoint, by minimizing the sum of the lengths of the edges joining the vertices. It means that the tripoint should be placed at the centroid of the triangle with $\mathbf{V}_{ij}, \mathbf{V}_{ik}$ and $\mathbf{V}_{jk}$ as vertices, that is:

$$\mathbf{V}_{ijk} = \frac{\mathbf{V}_{ij} + \mathbf{V}_{ik} + \mathbf{V}_{jk}}{3} \tag{3.4}$$

Where the two type-i corners are diagonally opposite, each cube edge around the face will has a separating vertex. It is an ambiguity that how the vertices should be connected to form elements of 1-skeleton. We present two reasonable ways at **Fig. 3.3(b)**. By the same arguments of the SC's approach (the choice of edges makes no topological different), we select the simpler case that constructs no interior vertices. We join the two $\mathbf{V}_{ij}$ by a single edge, and the two $\mathbf{V}_{ik}$ with another. The rule "don't cross the o–o diagonal" now becomes "don't cross the i–i diagonal".

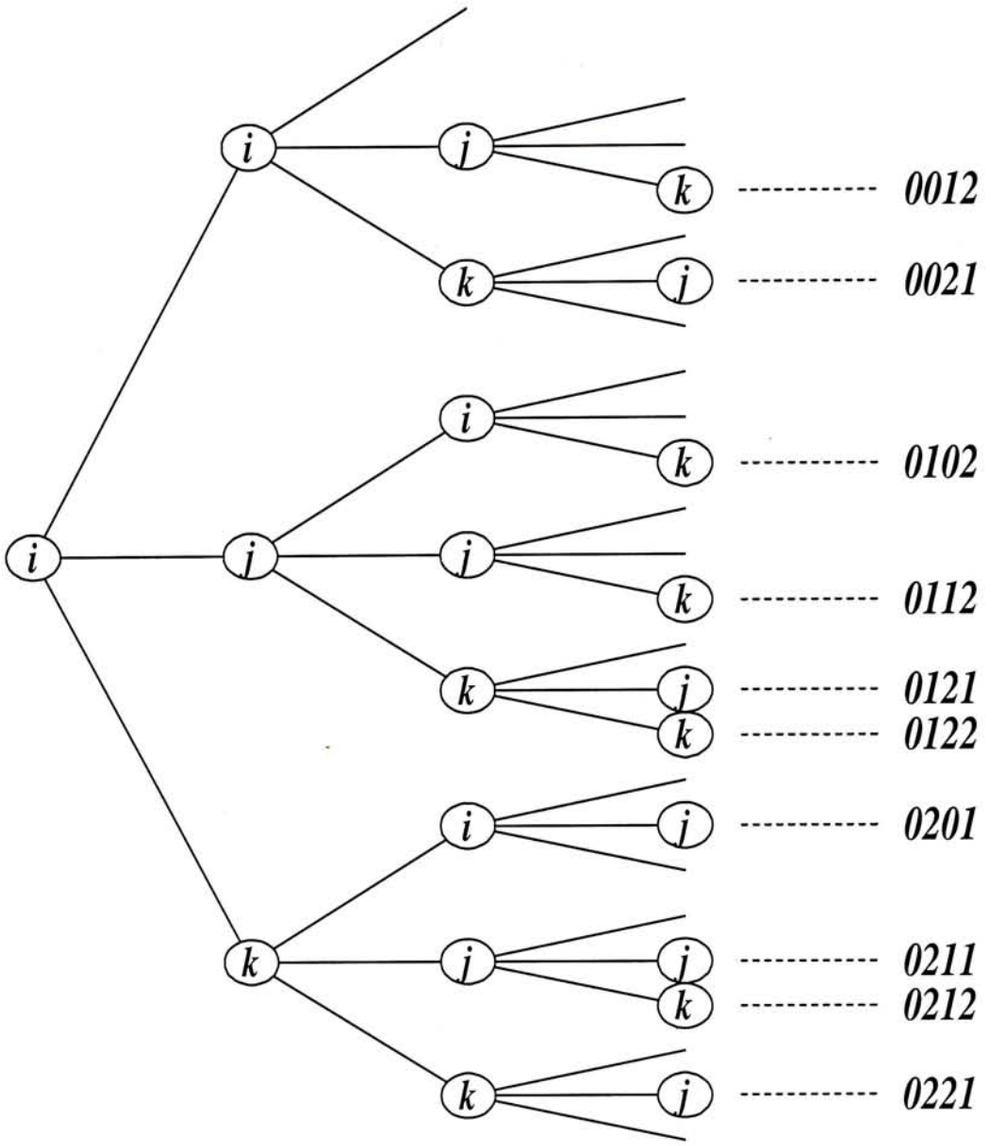


Figure 3.4: Ten basic patterns of three-type faces are obtained by rotational symmetry.

When we distribute the four samples with types i, i, j, k to the four corners of a cube face, there are $\frac{4!}{2!1!1!}$ ways. Since the repeated type may also be j or k, there are totally $(3 \times \frac{4!}{2!1!1!}) = 36$ number of cases of three-type faces. We can reduce the number of cases by considering the rotational symmetry, where the cube faces are rotated so that the first lowest type i is always at the lower-left corner, as shown in **Fig. 3.4**. As a result, there are ten basic patterns of three-type faces, and six cases with tripoints added, as shown in **Fig. 3.5(a)**, and four cases without tripoints, as shown in **Fig. 3.5(b)**. Similar to the binary faces, edges are created by joining the vertices, and are oriented to keep the higher type to the left.

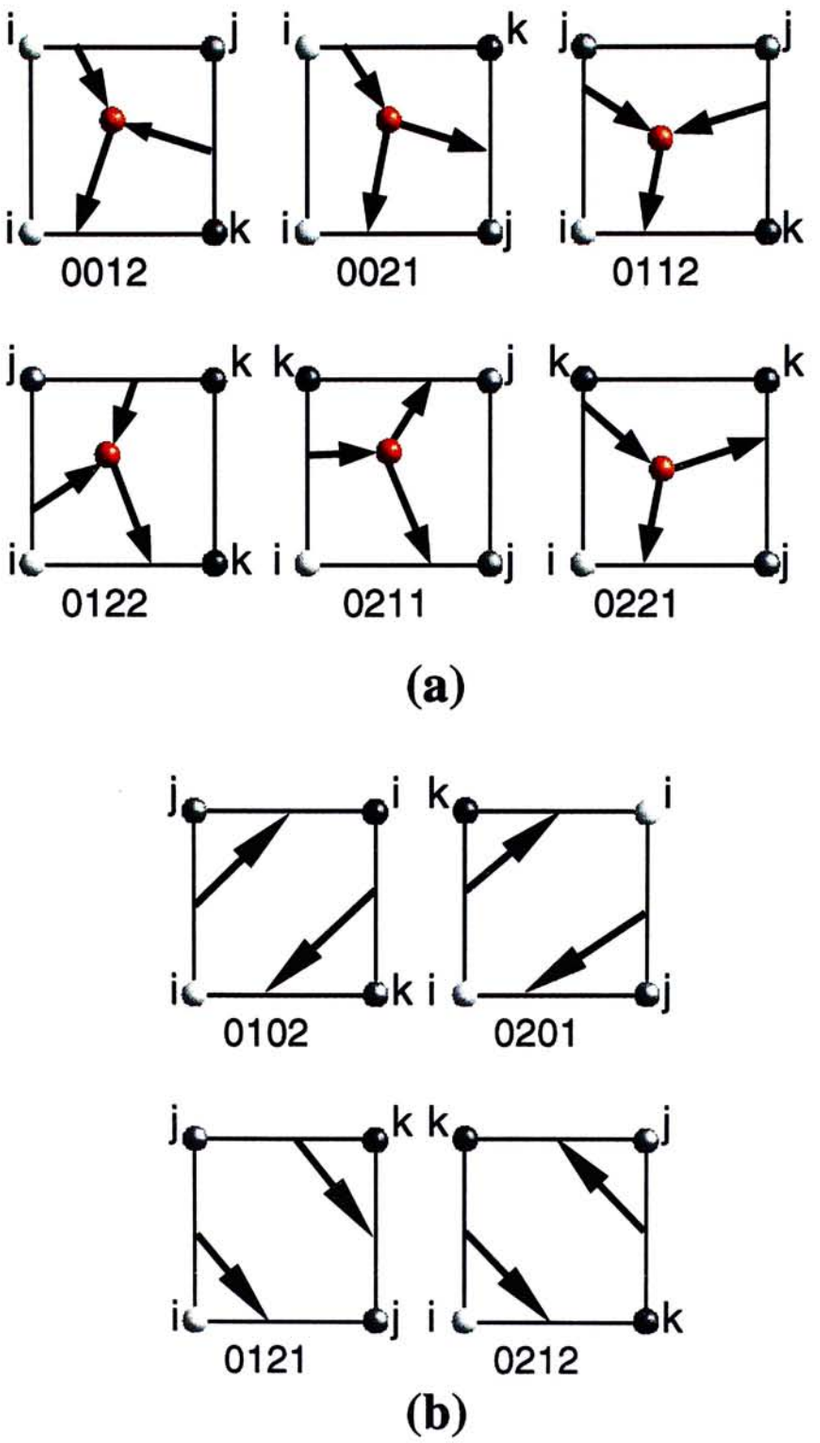


Figure 3.5: Ten cases of three-type faces: (a) With tripoints, (b) Without tripoints.

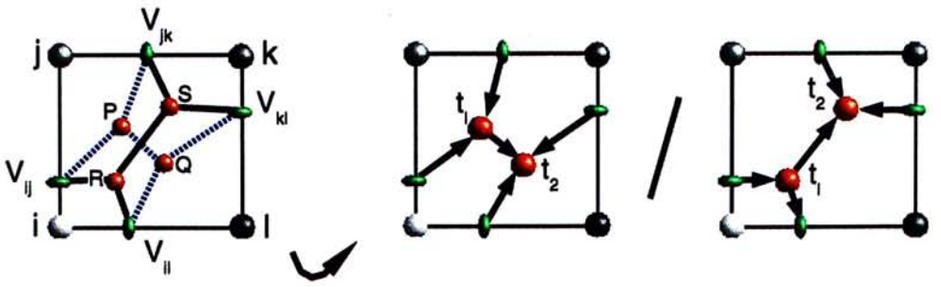


Figure 3.6: Four-type face topology – two ways to construct the tripoints.

Four-type faces

If four different types i, j, k, l are distributed among the four corners of a cube face, all the four edges around the face are *occupied*, as shown in **Fig. 3.6**, We find the separating vertices, $\mathbf{V}_{ij}$, $\mathbf{V}_{jk}$, $\mathbf{V}_{kl}$ and $\mathbf{V}_{il}$ at the corresponding edges, by linear interpolation using proper thresholds. We know that, when four types meet in 2D, there should be at least one point of contact of their boundaries. However, we can always replace the quadruple point by two tripoints without changing the value sampled at the four face corners. As **Fig. 3.6** shows, there are two ways to introduce the tripoints. First, we can select $\mathbf{P}, \mathbf{Q}$ as tripoints, where $\mathbf{P} = \mathbf{V}_{ijk}$ and $\mathbf{Q} = \mathbf{V}_{ikl}$. Second, we can also select $\mathbf{R}$, $\mathbf{S}$ as tripoints, where $\mathbf{R} = \mathbf{V}_{ijl}$ and $\mathbf{S} = \mathbf{V}_{jkl}$. We connect the corresponding edge vertices to the two tripoints and also join two tripoints together to form elements of 1-skeleton. By minimizing the sum of the edge lengths, we can decide which tripoints to be chosen and the way to construct edges.

Suppose, we have chosen $\mathbf{P}$,$\mathbf{Q}$ as the tripoints, and constructed the edges: $\mathbf{V}_{ij}\mathbf{P}$, $\mathbf{V}_{jk}\mathbf{P}$, $\mathbf{PQ}$, $\mathbf{V}_{kl}\mathbf{Q}$ and $\mathbf{V}_{il}\mathbf{Q}$. We can compute the positions of $\mathbf{P}$,$\mathbf{Q}$ as follows:

$$
\begin{aligned}
L &: \text{ Total length of the edges.} \\
L^2 &= (\mathbf{V}_{ij}-\mathbf{P})(\mathbf{V}_{ij}-\mathbf{P})^T + (\mathbf{V}_{jk}-\mathbf{P})(\mathbf{V}_{jk}-\mathbf{P})^T + (\mathbf{P}-\mathbf{Q})(\mathbf{P}-\mathbf{Q})^T + \\
&\quad (\mathbf{V}_{kl}-\mathbf{Q})(\mathbf{V}_{kl}-\mathbf{Q})^T + (\mathbf{V}_{il}-\mathbf{Q})(\mathbf{V}_{il}-\mathbf{Q})^T \qquad (3.5) \\
&\frac{\delta L^2}{\delta \mathbf{P}} = 0 \\
\Rightarrow\ & -2(\mathbf{V}_{ij}-\mathbf{P}) - 2(\mathbf{V}_{jk}-\mathbf{P}) + 2(\mathbf{P}-\mathbf{Q}) = 0
\end{aligned}
$$

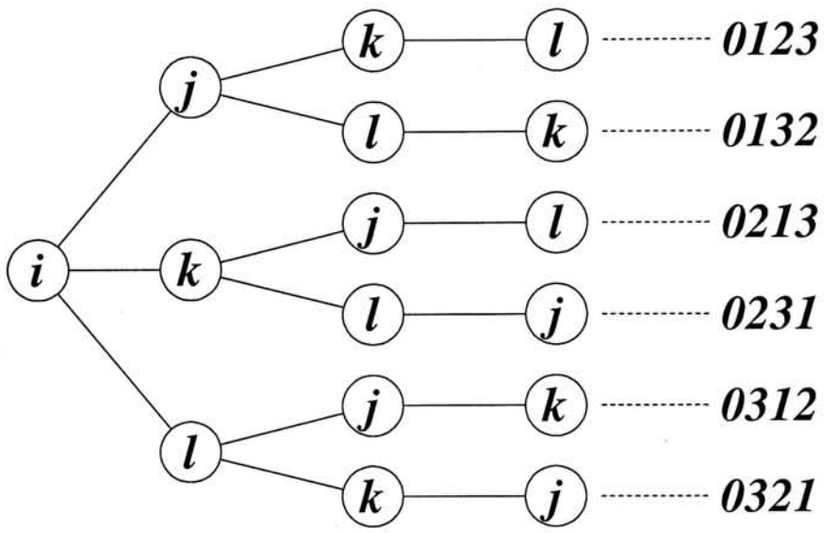


Figure 3.7: Six basic patterns of four-type faces are obtained by rotational symmetry.

$$\Rightarrow \quad \mathbf{P} = \frac{1}{3}(\mathbf{V}_{ij} + \mathbf{V}_{jk} + \mathbf{Q}) \tag{3.6}$$

$$\begin{aligned} & \frac{\delta L^2}{\delta \mathbf{Q}} = 0 \\ \Rightarrow \quad & -2(\mathbf{V}_{kl} - \mathbf{Q}) - 2(\mathbf{V}_{il} - \mathbf{Q}) - 2(\mathbf{P} - \mathbf{Q}) = 0 \\ \Rightarrow \quad & \mathbf{Q} = \frac{1}{3}(\mathbf{V}_{kl} + \mathbf{V}_{il} + \mathbf{P}) \end{aligned} \tag{3.7}$$

Solved **Eqn. 3.6** and **Eqn. 3.7**, we get:

$$\mathbf{V}_{ijk} = \mathbf{P} = \frac{3}{8}(\mathbf{V}_{ij} + \mathbf{V}_{jk}) + \frac{1}{8}(\mathbf{V}_{il} + \mathbf{V}_{kl}) \tag{3.8}$$

$$\mathbf{V}_{ikl} = \mathbf{Q} = \frac{3}{8}(\mathbf{V}_{il} + \mathbf{V}_{kl}) + \frac{1}{8}(\mathbf{V}_{ij} + \mathbf{V}_{jk}) \tag{3.9}$$

Similarly, if we have chosen **R**, **S** as the tripoints and construct the edges: $\mathbf{V}_{ij}\mathbf{R}$, $\mathbf{V}_{il}\mathbf{R}$, $\mathbf{RS}$, $\mathbf{V}_{jk}\mathbf{S}$ and $\mathbf{V}_{kl}\mathbf{S}$. The positions of **R**,**S** are computed:

$$\mathbf{V}_{ijl} = \mathbf{R} = \frac{3}{8}(\mathbf{V}_{ij} + \mathbf{V}_{il}) + \frac{1}{8}(\mathbf{V}_{jk} + \mathbf{V}_{kl}) \tag{3.10}$$

$$\mathbf{V}_{jkl} = \mathbf{S} = \frac{3}{8}(\mathbf{V}_{jk} + \mathbf{V}_{kl}) + \frac{1}{8}(\mathbf{V}_{ij} + \mathbf{V}_{il}) \tag{3.11}$$

Finally, we compare the total edge lengths:

$$\begin{aligned} & \mathbf{V}_{ij}\mathbf{P} + \mathbf{V}_{jk}\mathbf{P} + \mathbf{PQ} + \mathbf{V}_{il}\mathbf{Q} + \mathbf{V}_{kl}\mathbf{Q} \\ \leq \quad & \mathbf{V}_{ij}\mathbf{R} + \mathbf{V}_{il}\mathbf{R} + \mathbf{RS} + \mathbf{V}_{jk}\mathbf{S} + \mathbf{V}_{kl}\mathbf{S} \end{aligned} \tag{3.12}$$

and then use **P**, **Q** as the tripoints, if **Eqn. 3.12** is true and use **R**, **S** as the tripoints otherwise.

At a four-type face, there is only one combinatorial case with all the face corners are of different types. But with the different order of occurrence of the four types involved, there are $4! = 24$ cases of four-type faces, which can be reduced to six basic patterns by rotational symmetry, as shown in **Fig. 3.7**. However, since there are two ways of selecting the tripoints, we have totally twelve cases. **Fig. 3.8** shows the six cases using **P**, **Q** as tripoints. Similarly, we create elements of the 1-skeleton by joining the vertices along the face edges and tripoints inside the faces, and orient them to keep the higher type to the left.

3.3.2 Non-binary Cubes

After building the 1-skeleton elements on all faces, we have a graph on the surface of the cubes, with normal vertices on cube edges and tripoints inside faces. Our next step is to add 1-skeleton elements inside the cubes, so that the whole 1-skeleton found for the cubes can be expressed as an union of oriented loops (to be tessellated to form triangles, as will be discussed at **Section 3.5**).

It is possible that a non-binary cube has not tripoints, as shown in **Fig. 3.9**. In this case, it does not require adding of any extra vertices or extra edges inside the cube to form edge loops. When three types are involved at the cube corners and the cube has tripoints (all must have the same labels), we add only *tri-edges* to the cube that connect the tripoints to form *tricurves*. For more complex cases, it requires adding of both internal vertices (known as *tetrapoints*), and tri-edges joining them with tripoints or other tetrapoints.

Three-type cubes

Suppose we are given a cube in which the three types found at the corners are i, j, k with $i < j < k$, it is possible that no tripoints are introduced in the faces. In these cases, we can create edge loops by just connecting the oriented edges occur at the cube face. If there are tripoints, they occur in two forms. When viewed from outside the cube, the types of object meeting at it can be seen as in either positive-cyclic or negative-cyclic order, as shown in **Fig. 3.10**. The corresponding tripoints are called *positive* or *negative tripoint*. Note that the 'higher type at the left' rule orients (i, j) and (j, k) edges outward from the positive tripoints, (i, k) edge inwards, and *vice versa* for a negative tripoint. Therefore, a positive tripoint has two outgoing edges and one incoming edge, while a negative tripoint has one outgoing edge and two incoming edges Since the oriented edges and the vertices on the cube faces formed a directed graph,

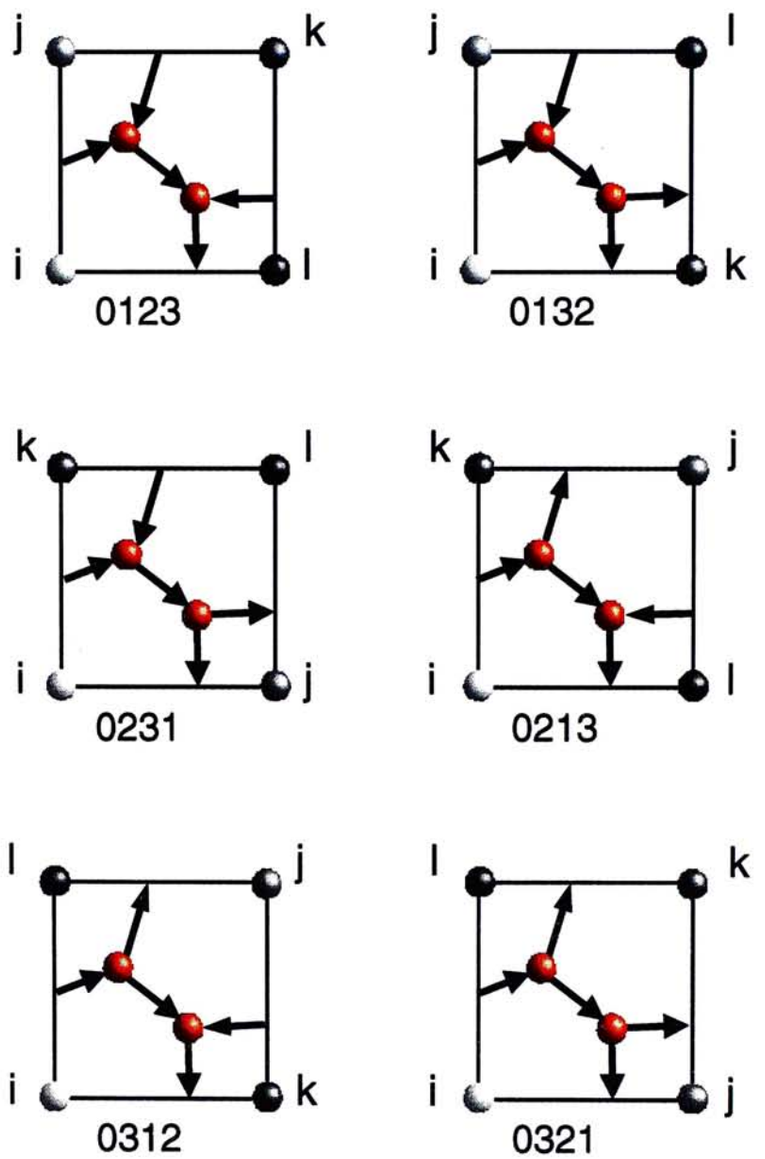


Figure 3.8: Six cases of four-type faces.

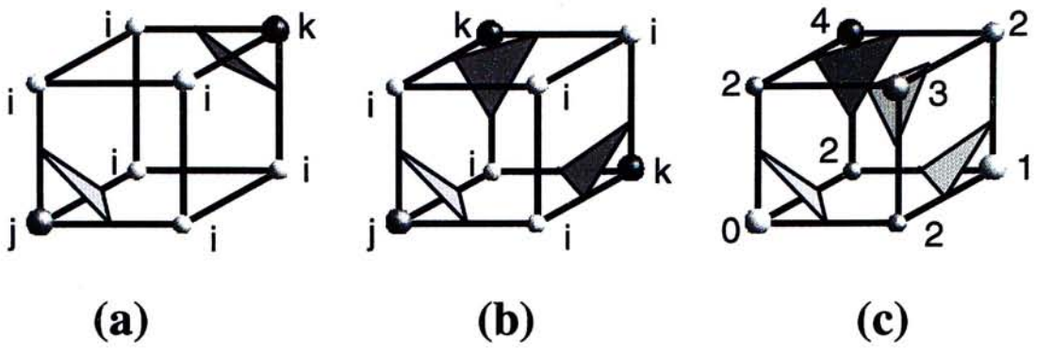


Figure 3.9: It is possible that non-binary cubes have not tripoints, such as: (a) A three-type cube with all binary faces, (b) A three-type cube with non-binary faces, (c) A messy cube with five types.

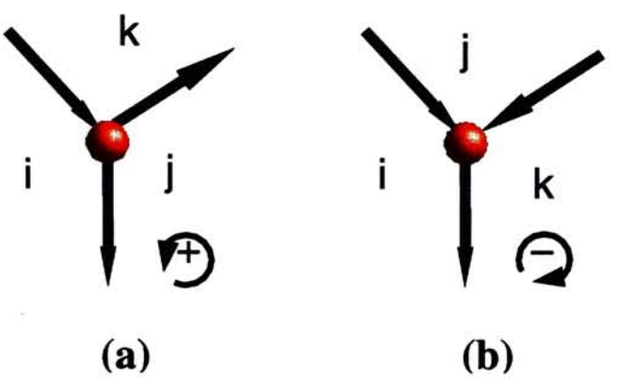


Figure 3.10: Tripoints occur in two forms: (a) Positive, (b) Negative.

the two types of tripoints are necessarily occur in equal numbers on the surface of a given cube (for the proof, please refer to **Appendix A**). We create (i, j, k) tri-edges interior to the cube, joining positive and negative tripoints in pairs. Each tri-edge joining two tripoints can be regarded as a bundle of three edges (namely, (i, j), (j, k) and (i, k)). The orientation of the tri-edge is that, it goes from negative tripoint to positive tripoint when it is thought of as an (i, j) or (j, k) edge, and goes from positive tripoint to negative tripoint when it is thought of as an (i, k) edge. The orientations are necessary for the creation of oriented edges loops.

If there is only one pair of tripoints, there is only one way to create a tri-edge. Where there are four tripoints, there are (bearing in mind the requirement of joining positive to negative) only two ways to join them. Where there are six tripoints, there are six ways to join them. Since for a three-type face we never put more than one tripoint on a face, this is the most complex situation. Where there is a choice, we select the way of joining them that minimizes total edge lengths.

Four-type cubes

Four-type cubes have four different types i, j, k, l involved at the cube corners, where $i < j < k < l$. It is also possible that no tripoints are introduced in the faces. However, if it does have tripoints, there must be at least four tripoints $\mathbf{V}_{ijk}$, $\mathbf{V}_{ijl}$, $\mathbf{V}_{ikl}$ and $\mathbf{V}_{jkl}$ on the cube surface. If it has exactly four, we need a single tetrapoint inside the cube:

$$\mathbf{V}_{ijkl} = \frac{\mathbf{V}_{ijk} + \mathbf{V}_{ijl} + \mathbf{V}_{ikl} + \mathbf{V}_{jkl}}{4} \tag{3.13}$$

Then we connect $\mathbf{V}_{ijkl}$ to the tripoints by four tri-edges of labels, (i, j, k), (i, j, l), (i, k, l) and (j, k, l). The (i, j, k) tri-edge connects $\mathbf{V}_{ijkl}$ to $\mathbf{V}_{ijk}$, and orients such that: if $\mathbf{V}_{ijk}$ is a negative tripoint, it goes from $\mathbf{V}_{ijk}$ to $\mathbf{V}_{ijkl}$ when it is thought of as an (i, j) or (j, k) edges, and goes from $\mathbf{V}_{ijkl}$ to $\mathbf{V}_{ijk}$ when it is thought of as an (i, k) edge. If there are more than four tripoints on the surface of a cube with four types at its corners, one tetrapoint may not be enough. We handle these cases by the general scheme below.

3.4 General Scheme for Messy Cubes

A messy cube has more than three types of samples occur at its eight corners, and has more than four tripoints. To create edge loops for the messy cube, more than one tetrapoint is required to be added inside the cube. In this section, we present a general scheme that generates a minimum number of tetrapoints and minimizes the sum of the lengths of the 1-skeleton elements. (**Appendix B** is a supplement to this section. Readers can refer to it for an example of isosurface extraction of a messy cube.) Before we begin the discussion, we first define:

K – the labeled, oriented 1-skeleton elements on the cube faces that form a directed graph $= (U, E)$.

U – set of the vertices.

E – set of the oriented edges.

W – set of the tripoints, thus $W \subseteq U$.

The process begins with a construction of an initial graph, G. The initial set of vertices of G is W, the tripoints of K. Two vertices of G are joined by an initial edge e, if and only if, in K they are joined by a path that meets no other tripoints. The idea of the process is that we add tetrapoints and tri-edges to K, based on the structure of

G. When the process goes on, we reduce the graph G, replacing it with a new graph with lesser vertices and edges. As a result, when G becomes empty, we have added a minimum number of tetrapoints and tri-edges to K. We then assign the positions of the tetrapoints by minimizing the sum of the lengths of the internal edges.

3.4.1 Graph Reduction

We first try to look for the tripoints with same labels. We removed those tripoints and the corresponding edges from G, but add a tri-edge joining them to E. If we cannot find tripoints with same labels, we look for loops of three or four tripoints. We delete the corresponding loops and edges, but add new vertices and edges to both G and K. The new vertices added including tripoints and tetrapoints, but we will remove tripoints later after merging the edges joining them with other vertices. The new edges added are two types of tri-edges. They are tri-edges joining two tripoints and tri-edges joining a tripoint with a tetrapoint. The tri-edges joining two tetrapoints can be thought as a merging of two tri-edges joining the tetrapoints with a tripoint The tri-edges are labeled and oriented consistently, depending on the labels and types of its tripoints. There are five possible changes that we can apply to G, as shown in **Fig. 3.11**. (In **Appendix A**, we will prove that it is always possible to apply at least one of the changes, until G becomes empty.)

Case (a), (b) and (c) :

In these three cases, we look for tripoints of same labels, $\{i, j, k\}$. As shown in **Fig. 3.11(a)**, **(b)** and **(c)**, we add only a tri-edge to E joining the tripoints **p** and **q**. We replace the subgraph of G (consisting of **p**, **q** and edges incident on them), with edges joining the vertices that linked to **p**, **q** originally. The number of edges we add to G depend on how many edges connecting **p** and **q** together. In **(a)**, **p** and **q** are joined by three edges. Since each tripoint has only three edges incident on it, this subgraph of G is isolated. We then simply eliminate the subgraph from G. In **(b)**, **p** and **q** are joined by exactly two edges. We change G by replacing the subgraph with a new edge joining the vertices **P** and **Q**. In **(c)**, **p** and **q** are joined by only one edge, we change G by replacing the subgraph with two edges, joining **P** with **Q** and **R** with **S**.

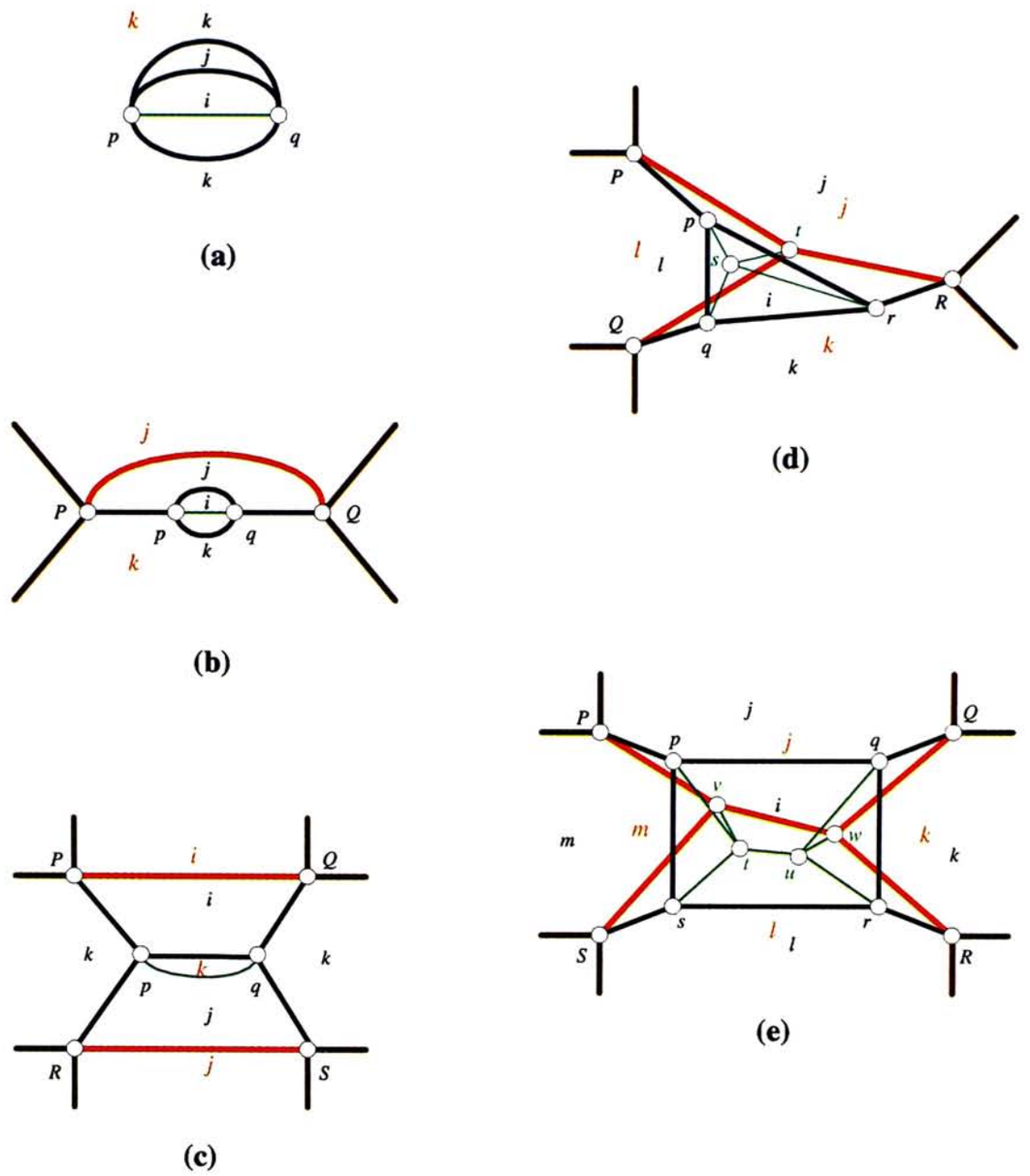


Figure 3.11: Changes in G, the subgraph shown in black is replaced by the subgraph shown in red, and the edges and vertices shown in green are added to K.

Case (d) :

If there is no place where we can apply one of **(a)**, **(b)** or **(c)**, we look for the shortest loop L in G containing exactly three vertices (call them **p**, **q**, **r**) and apply the changes in **Fig. 3.11(d)**. Since the new vertices will not have (x, y, z) coordinates attached until computed later, the term 'shortest' will be understood as follow. Where edges have ends in the original W, we compare their geometric length. An edge with both ends new is 'shorter' than any edges with only one, and an edge with only one is 'shorter' than any edges with both ends in W. If there is a tie, we break it arbitrarily. We add to U a tetrapoint **s** and a tripoint **t**, and add to E four tri-edges joining **s** to **p**, **q**, **r** and **t**. The tripoint **t** is also added to G, with edges that join it to the not-in-L vertices **P**, **Q** and **R** that originally linked to **p**, **q** and **r** respectively.

Case (e) :

If G has no instance of **(a)**, **(b)**, **(c)** or **(d)**, we look for the shortest loop L in G containing exactly four vertices (call them **p**, **q**, **r**, **s**) and apply the changes in **Fig. 3.11(e)**. We add to U two tetrapoints **t**, **u** and two tripoints **v**, **w** and add to E seven tri-edges joining **t** to **p**, **s**, **v**, and **u** to **q**, **r**, **w**, and joining **t**, **v** together. The tri-edge joining **t** and **v** can be thought as a merging of two tri-edges joining them to an implicit tripoint of label (i, j, l). The tripoints **v**, **w** are also added to G, with edges that join them to the no-in-L vertices **P**, **Q**, **R**, **S** and join them together. There are two versions of these changes, since we could link the new tetrapoints to the pairs $\{\mathbf{p}, \mathbf{q}\}$ and $\{\mathbf{r}, \mathbf{s}\}$ instead of $\{\mathbf{q}, \mathbf{r}\}$ and $\{\mathbf{p}, \mathbf{s}\}$. We decide the way of changes by minimizing the total edge lengths. If the total distances between $\{\mathbf{p}, \mathbf{q}\}$ and between $\{\mathbf{r}, \mathbf{s}\}$ is shorter than the total distance between $\{\mathbf{q}, \mathbf{r}\}$ and between $\{\mathbf{p}, \mathbf{s}\}$, we should use the other way not shown.

3.4.2 Position of the Tetrapoints

We apply the above reduction steps repeatedly to G until it becomes empty. After the process, we have added new tripoints and tetrapoints, and edges connecting them to K. Each newly added tripoints should have exactly two internal edges incident on it. These two edges are of matching labels, meaning that they connect the newly added tripoint to either other tripoints of same labels, or tetrapoints whose labels are supersets of the label of the tripoint. Therefore, we can always delete a new tripoint, merging two edges connected to it, until only T tetrapoints remain of the vertices we

have added to U. We now have K modified to K' with the tripoints on the surface of the cube and tetrapoints somewhere inside the cube. The next step is to assign the (x, y, z) positions to the T tetrapoints by minimizing the sum of the edge lengths of the internal edges in K' that join them to tripoints and to one another.

To illustrate the computation of the positions of the tetrapoints, let us consider the following example. Suppose we have added 2 tetrapoints $(\mathbf{t}_1, \mathbf{t}_2)$, to a messy cube with six tripoints $(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}, \mathbf{E}, \mathbf{F})$ on its faces. Moreover, we also have added tri-edges joining $\mathbf{t}_1$ to $\mathbf{A}, \mathbf{B}, \mathbf{C}$, and $\mathbf{t}_2$ to $\mathbf{D}, \mathbf{E}, \mathbf{F}$, and joining $\mathbf{t}_1, \mathbf{t}_2$ together. The positions of $\mathbf{t}_1$ and $\mathbf{t}_2$ are computed:

$$
\begin{aligned}
L &: \text{ Total length of the internal edges.} \\
L^2 &= (\mathbf{t}_1 - \mathbf{A})(\mathbf{t}_1 - \mathbf{A})^T + (\mathbf{t}_1 - \mathbf{B})(\mathbf{t}_1 - \mathbf{B})^T + (\mathbf{t}_1 - \mathbf{C})(\mathbf{t}_1 - \mathbf{C})^T + \\
&\quad (\mathbf{t}_2 - \mathbf{D})(\mathbf{t}_2 - \mathbf{D})^T + (\mathbf{t}_2 - \mathbf{E})(\mathbf{t}_2 - \mathbf{E})^T + (\mathbf{t}_2 - \mathbf{F})(\mathbf{t}_2 - \mathbf{F})^T + \\
&\quad (\mathbf{t}_1 - \mathbf{t}_2)(\mathbf{t}_1 - \mathbf{t}_2)^T \qquad (3.14)
\end{aligned}
$$

$$
\begin{aligned}
&\frac{\delta L^2}{\delta \mathbf{t}_1} = 0 \\
\Rightarrow\ & 2(\mathbf{t}_1 - \mathbf{A}) + 2(\mathbf{t}_1 - \mathbf{B}) + 2(\mathbf{t}_1 - \mathbf{C}) + 2(\mathbf{t}_1 - \mathbf{t}_2) = 0 \\
\Rightarrow\ & 4\mathbf{t}_1 - \mathbf{t}_2 = \mathbf{A} + \mathbf{B} + \mathbf{C} \qquad (3.15)
\end{aligned}
$$

$$
\begin{aligned}
&\frac{\delta L^2}{\delta \mathbf{t}_2} = 0 \\
\Rightarrow\ & 2(\mathbf{t}_2 - \mathbf{D}) + 2(\mathbf{t}_2 - \mathbf{E}) + 2(\mathbf{t}_2 - \mathbf{F}) - 2(\mathbf{t}_1 - \mathbf{t}_2) = 0 \\
\Rightarrow\ & 4\mathbf{t}_2 - \mathbf{t}_1 = \mathbf{D} + \mathbf{E} + \mathbf{F} \qquad (3.16)
\end{aligned}
$$

Rewrite **Eqn. 3.16** and **Eqn. 3.16** in matrix form, we have:

$$
\begin{pmatrix} 4 & -1 \\ -1 & 4 \end{pmatrix} \begin{pmatrix} \mathbf{t}_1 \\ \mathbf{t}_2 \end{pmatrix} = \begin{pmatrix} \mathbf{A} + \mathbf{B} + \mathbf{C} \\ \mathbf{D} + \mathbf{E} + \mathbf{F} \end{pmatrix} \qquad (3.17)
$$

Therefore, to find the (x, y, z) coordinates of the 2 tetrapoints, we need to solve three 2×2 matrices in x, y and z coordinates separately. Similarly, to find the positions of T tetrapoints is a linear problem of solving three $T \times T$ matrices.

3.5 Triangular Mesh Generation

Once we have created the 1-skeleton, with oriented edges on the cube faces joining normal vertices and tripoints, and tri-edges inside the cubes joining tripoints and

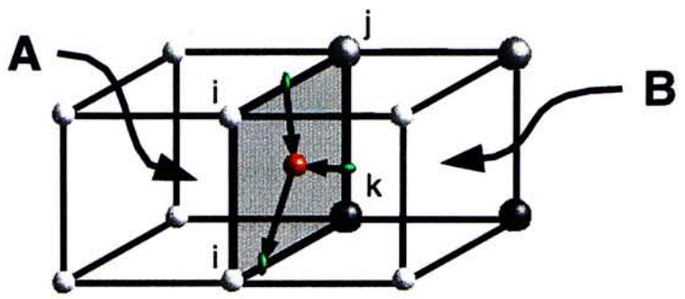


Figure 3.12: Edges on a shared face should be reversed, when we construct edges loops for cube B.

tetrapoints, we are ready to generate the 2-skeleton (isosurfaces approximated as triangular mesh) and the 3-skeleton (the multi-body surface). We first generate the edges loops by connecting the 1-skeleton elements. Then, tessellate the edge loops to form triangular mesh by 'nibbling' process. A multi-resolution surface can be generated by incorporating with the Adaptive Skeleton Climbing algorithm.

3.5.1 Generating the Edge Loops

For binary cubes, there are only one type of edge loops with label (i, j). We create the edges loops just as SC does by joining the 1-skeleton elements. For non-binary cubes, there are at most $\frac{n(n-1)}{2}$ types of edge loops, where $8 \geq n \geq 3$ is the number of types occur at eight cube corners. For example, a three-type cube with types i, j, and k has three types of edge loops: (i, j), (j, k) and (i, k). We create the edges loops by connecting the edges with same labels, such that:

- Edges on cube faces occur exactly once.
- Tri-edges strictly inside cubes occur three times.

Note that, in order to construct the loops, the oriented edges on a cube face shared by two adjacent cubes should be properly reversed, as shown in **Fig. 3.12**.

Every edge loop in binary or three-type cubes must at least contains one normal vertex that lies on a cube edge. Therefore, we can create all the edge loops of these cubes by starting from every occupied edge. For every edge vertex that is*not visited* in the previous edge loops, we follow the oriented edge incident on it and go to the next vertex. The next vertex can be either a normal vertex or a tripoint (if the cube is a three-type cube). If it is a normal vertex, we just follow its outgoing edges and go

to the next vertex. If it is a tripoint, we follow the corresponding orientation of the tri-edge and go to another tripoint. The process is repeated until we reach the starting vertex. The edge loop created is labeled and put into the output list.

For cubes with more than three-types, we may have loops that do not contain normal vertices. In a messy cube, we can have as many as three types of edge loops: 1) edge loops involving all types of vertices, 2) edge loops involving only tripoints and tetrapoints, 3) edge loops involving only tetrapoints. The first type of edge loops is created as usual, starting with edge vertices. We generate an edge loop of the second type by starting with the two tripoints in a four-type face. The label of the edge loop is the intersection of the labels of the tri-points. Consider a four-type face with two tripoints, the edge connecting the tripoints separates the two diagonally opposite corners. Therefore, the edge loop will be missed, if we generate the edge loops by starting with edge vertices. The third type of edge loops involving only tetrapoints, and must exist only inside messy cubes. They separate two corners of a cube with the longest distance. These edge loops will be missed, if we only generate the edge loops by starting with either normal vertices at cube edges, or tripoints inside cube faces. They are created by examining if there are loops existed in the tetrapoints we added to a messy cube by the general scheme. A loop formed by a set of tetrapoints is consider being an edge loop, if it is the smallest loop that does not contain any sub-loops. The label of the edge loops is the intersection of the labels of the tetrapoints. The edge loops of different types are generated in order, as the algorithm `GenLoops` shown in **Fig. 3.13**, and then put into the output list.

As mentioned before, a tripoint has three distinct normal vectors and a tetrapoint has six. We cannot use central differences to compute these normal vectors as it only approximates a unique normal vector of a point in a volumetric data based on the gradient of the sampled value. By the assumption that the normal vectors of the vertices in the same edge loop will not deviate largely in direction, we can approximate some of the normal vectors of a tripoint or a tetrapoint by other vertices in the same edge loops. For a tripoint in an edge loop containing normal vertices, the corresponding normal vector (of types of the edge loop label) *inherit* the normal vector of the immediate preceding vertex. This allows all the three normal vectors of a tripoint in a three-type cube to be approximated, and allows two of the normal vectors of a tripoint in a four-type face to be approximated. The other normal vector of the tripoints in four-type faces can then be computed, as the vector sum of the normal vectors of a point should be zero. After all the normal vectors of tripoints are computed, the normal vectors of tetrapoints are approximated similarly by first inherit the normal vectors of vertices in the same edge loop. Then, the remaining normal vector of a tetrapoint is computed to

---

Input: A set of vertices, U
A set of oriented edges, E
Output: A set of labeled edge loops, L
Algorithm: Connecting oriented edges to form edge loops

$N \subseteq U$ /* the set of the normal vertices */
$W \subseteq U$ /* the set of the tripoints */
$T \subseteq U$ /* the set of the tetrapoints */

/* Loops involving all types of vertices */
FOR every occupied edge **DO**
  IF $\mathbf{V}_s \in N \wedge \mathbf{V}_s.visited =$ FALSE **THEN**
    $l \leftarrow \{\mathbf{V}_s\}$, $l.type \leftarrow \mathbf{V}_s.type$
    $\mathbf{V}_s.visited \leftarrow$ TRUE, $\mathbf{V} \leftarrow \mathbf{V}_s$
    WHILE $\mathbf{I} \neq \mathbf{V}_s \wedge (\mathbf{V},\mathbf{I}) \in E \wedge l.type = Type((\mathbf{V},\mathbf{I}))$ **DO**
      $\mathbf{I}.visited \leftarrow$ TRUE, $\mathbf{V} \leftarrow \mathbf{I}$, $l \leftarrow l \cup \{\mathbf{I}\}$
    ENDWHILE
    $L \leftarrow L \cup \{l\}$
  ENDIF
ENDFOR

/* Loops involving only tripoints and tetrapoints */
FOR every four-type face **DO**
  IF $\mathbf{V}_s, \mathbf{X}_s \in W \wedge (\mathbf{V}_s, \mathbf{X}_s) \in E$ **THEN**
    $l \leftarrow \{\mathbf{V}_s, \mathbf{X}_s\}$, $l.type \leftarrow \mathbf{V}_s.type \cap \mathbf{X}_s.type$
    $\mathbf{V}_s.visited \leftarrow \mathbf{X}_s.visited \leftarrow$ TRUE, $\mathbf{V} \leftarrow \mathbf{X}_s$
    WHILE $\mathbf{I} \neq \mathbf{V}_s \wedge (\mathbf{V},\mathbf{I}) \in E \wedge l.type = Type((\mathbf{V},\mathbf{I})) \wedge \mathbf{I} \notin N$ **DO**
      $\mathbf{I}.visited \leftarrow$ TRUE, $\mathbf{V} \leftarrow \mathbf{I}$, $l \leftarrow l \cup \{\mathbf{I}\}$
    ENDWHILE
    $L \leftarrow L \cup \{l\}$
  ENDIF
ENDFOR

/* Loops involving only tetrapoints */
FOR every loop $l_t = \{\mathbf{t}_1, \mathbf{t}_2, ..., \mathbf{t}_n\} \wedge \mathbf{t}_i \in T$ **DO**
  $l.type \leftarrow \mathbf{t}_1.type \cap \mathbf{t}_2.type \cap, ..., \cap \mathbf{t}_n.type$
  $L \leftarrow L \cup \{l_t\}$
ENDFOR
Return L

Figure 3.13: Algorithm `GenLoops` for generation of edge loops of a messy cube.

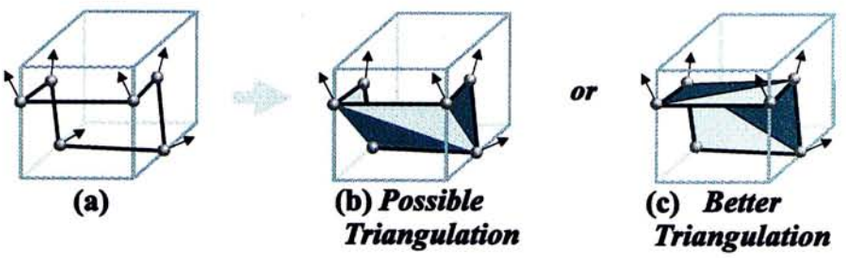


Figure 3.14: Triangulate the edge loop to emit triangles.

be the negation of the sum of the inherited normal vectors. The order of the generation of the edge loops, as shown in **Fig. 3.13** is the fundamental of this inheritance of the normal vectors.

3.5.2 Triangulating the Edge Loops

Given an edge loop consisting of several vertices, $\mathbf{V}_1$, $\mathbf{V}_2$, ..., $\mathbf{V}_n$, we emit triangles using the algorithm `EmitTriangle` in **Fig. 3.15**. In each iteration, three consecutive vertices $\mathbf{V}_i$, $\mathbf{V}_{i+1}$ and $\mathbf{V}_{i+2}$ are selected, and one triangle is generated. The fitness of the triangle is first evaluated, before putting the triangle into the output list. An edge loop can be triangulated in multiple ways. Different sequences give triangular mesh with identical triangle counts, but with different geometry (**Fig. 3.14(b)** and **(c)**). To generate a mesh that closely approximates the true isosurface, we make use of the gradient. As shown in algorithm `EmitTriangle` in **Fig. 3.15**, we reject any triangle with planar normal vector $\mathbf{N}$ that largely deviates from the gradients $\mathbf{G}_i$ at three vertices. The deviation is measured by the dot product of $\mathbf{N}$ and $\mathbf{G}_i$. A threshold is used as a criterion. The threshold constraint will be relaxed if no triangle can be generated under the current constraint. After the triangle is put into the output list, The vertex $\mathbf{V}_{i+1}$ is then removed from the edge loop. The algorithm continues until only two vertices are left.

The output of a single-body surface extraction algorithm is always a single boundary isosurface that separates the object and background. Though even the output of the multi-body surface extraction algorithm may also be a single surface (only when number of types is equal to two), the result of triangulation is generally a set of surface pieces. A surface piece, Σ_{ij}, separates type i in the positive direction of the vertices' normal, from type j in other direction, where $j > i$. The multi-body surface is the union of these isosurfaces. It can then be rendered for visualization or haptic interaction, with different materials are assigned to different isosurfaces. Note that the isosurface Σ_{ij}, are rendered only once, and thus a lot of computational resources are

---

Input: An edge loop, $L = \{\mathbf{V}_1, \mathbf{V}_2, ..., \mathbf{V}_n\}$
An initial deviation threshold, T
Output: A triangular mesh, M
Algorithm: Triangulating the edge loop by 'nibbling' process

$M = \emptyset$
WHILE $Number(L) > 2$ **DO**
    /* Pick three consecutive vertices from L, $C \subseteq L$ */
    $C \leftarrow \{\mathbf{V}_i, \mathbf{V}_{i+1}, \mathbf{V}_{i+2}\}$
    $X \leftarrow \triangle_{\mathbf{V}_i, \mathbf{V}_{i+1}, \mathbf{V}_{i+2}}$

    $\mathbf{N} \leftarrow NormalOfTriangle(X)$
    $\mathbf{G}_i \leftarrow Normal(\mathbf{V}_i)$
    $\mathbf{G}_{i+1} \leftarrow Normal(\mathbf{V}_{i+1})$
    $\mathbf{G}_{i+2} \leftarrow Normal(\mathbf{V}_{i+2})$

    IF $\mathbf{N} \bullet \mathbf{G}_i > T \vee \mathbf{N} \bullet \mathbf{G}_{i+1} > T \vee \mathbf{N} \bullet \mathbf{G}_{i+2} > T$ **THEN**
        Reject X
    ELSE
        $M \leftarrow M \cup \{X\}$
        $L \leftarrow L - \{\mathbf{V}_{i+1}\}$
    ENDIF

    IF no triangles can be generate using T **THEN**
        $T \leftarrow T - \delta$
    ENDIF
ENDWHILE
Return M

Figure 3.15: Algorithm `EmitTriangle` for triangulating of edge loops.

saved. Using these isosurfaces, we can also obtain a particular isosurface of type i easily. A surface separates type i from other others is:

$$\Sigma_i = \left(\bigcup_{k=0}^{i-1} -\Sigma_{ik} \right) \bigcup \left(\bigcup_{k=i+1}^{N} \Sigma_{ik} \right) \tag{3.18}$$

where N is the number of the types, and $-\Sigma_{ij}$ is obtained from Σ_{ij} by flipping the normal vector of each vertex.

3.5.3 Incorporating with Adaptive Skeleton Climbing

Just like the original MC algorithm [24], the multi-body surface extraction algorithm is a unit-sized partitioning approach. It subdivides the volumetric data into unit-sized cubes (composes of eight neighbor samples) and then generates triangles within the cubes. This approach has the problem of generating enormous number of triangles that are more than enough even for single isosurface, needless to say for a multi-body surface. ASC algorithm [34] generates triangles adaptively to the structure of the isosurface, from non-uniform sized blocks. When the enclosed isosurface in each block is smooth enough to be approximated by a larger triangle, it will not generate many small triangles as SC does. Therefore, a large number of triangles are saved. Moreover, unlike the triangle reduction algorithms, ASC algorithm generates coarser mesh directly from the original volumetric data. This ensures no distortion or error is introduced before the triangle reduction. More importantly, the algorithm is an on-the-fly process, which requires no time-consuming post-processing triangle reduction. In fact, the algorithm produces coarser mesh in a smaller amount of time [34]. Although the approach may not reduce triangles as much as mesh optimizer does, it is a cost-effective method to significantly reduce triangles in a short period of time.

There are not significant differences between generating isosurface from blocks and generating isosurface from unit-sized cubes. Therefore, the most important step of ASC is actually the partitioning of the volumetric data into non-uniform sized *simple* blocks. The similar technique can also be used in our algorithm to generate multi-body surfaces in multi-resolutions. However, although the building of simple highrices of more than two types is theoretically possible, it will certainly increase the complexity of the process. Moreover, it may require placing a lot of tripoints on the highrices surface, and thus require placing a lot of tetrapoints inside highrices. The computation of the positions of the tetrapoints become complex and it is less sure to generate non-self-intersecting isosurfaces inside messy cubes. Most importantly, there are usually a small percentage of regions where more than two types met in the volumetric data.

Therefore, instead of applying the ASC technique to the whole volumetric data, we only allow non-binary cubes to be combined to form simple highrices. This limited version of less adaptive when at or close to non-binary cubes, will greatly simplify the process with the sacrifice of generating a little bit more triangles.

The whole volumetric data is divided into cubic blocks of $(N+1)\times(N+1)\times(N+1)$ samples. The blocks are classified into binary or non-binary, depending on the number of types of the samples within each block. We then apply the ASC algorithm to binary blocks and applying the multi-body surface extraction algorithm to non-binary blocks. For binary blocks, we perform the volume partitioning process, and generate largest sized simple highrices. Recall that gaps will appear if no information is shared between adjacent highrices. Similar cracks will appear if information is not shared between adjacent blocks. Unlike the case of variable-sized highrices, each block has the same size. This simplifies the process. This information sharing process must be done just after the information sharing among highrices. For a non-binary block, we do not actually perform the volume partitioning. Thus, we direct create the data structures that represents the division of the block into the smallest unit-sized cubes, and then carry out the information sharing process between adjacent blocks.

The block size (the value of N) will not only constrain the maximum size of the highrices in binary blocks, but also determine the chance of getting non-binary blocks. Both of these will then decide the number of triangles generated. When the block size is small, say $N = 1$, the largest highrice contains $2 \times 2 \times 2$ samples, and it is actually a cube size. Therefore, smaller triangles are formed and thus a larger number of triangles are generated. When a larger block size is used, larger highrices are allowed to be generated, hence smaller number of triangles. However, we also will have a greater chance to get non-binary blocks. Since we subdivide non-binary blocks into smallest sized cubes, and generate isosurface from it. More non-binary blocks means lesser adaptive at more regions, and thus generates more triangles. In other words, increasing N does not necessary decrease the triangle counts. In fact, a larger N (thought allow larger triangles), generates more triangles in some cases. By controlling the value N, we can generate the multi-body surfaces in multiple resolutions. It is however, we must note that the parameter N is just an indirect control, and the actual mesh generated will also depend on the geometry of the real isosurface. More triangles will still be generated if the isosurface geometry is complex.

Volumetric Data	Number of Types	CPU Time (seconds)	Number of Triangles	Percentage of Non-Binary Cubes
spheres ($128 \times 128 \times 128$)	10	$24.15s$	252754△	0.00%
4types ($128 \times 128 \times 128$)	4	$21.52s$	185012△	0.05%
7types ($128 \times 128 \times 128$)	7	$23.46s$	237864△	0.16%
head2 ($256 \times 256 \times 98$)	3	$70.72s$	691520△	0.023%
cthead ($256 \times 256 \times 113$)	2	$66.08s$	503813△	0.00%
tomato ($128 \times 128 \times 64$)	6	$16.48s$	357943△	1.24%
orange ($128 \times 128 \times 64$)	7	$14.82s$	334945△	1.88%
frog ($128 \times 128 \times 136$)	16	$27.03s$	458806△	0.94%

Table 3.1: Quantitative results of Multi-body Surface Extraction algorithm with various volumetric data.

3.6 Implementation and Results

Table 3.1 is the quantitative result of our implementation of the multi-body surface extraction algorithm to various volumetric data. We show the triangle counts and CPU times on a SUN Ultra 5/270 workstation with 128Mb main memory and running Solaris 2.6 as OS. Also shown in the table are the number of types defined and percentage of the non-binary cubes in the volumetric data.

The "spheres" data is sampled from a spherical volume with the density increased towards the center. The resolution of the data is $128 \times 128 \times 128$, and 10 types of objects are defined. The multi-body surface is displayed in the **Fig. 3.18**. The isosurfaces extracted are concentric spherical surfaces. Though applied the algorithm to the "spheres", shows that the algorithm is able to extract the multi-body surface in a single step of processing, it does not tell us the performance of the algorithm to the places where more than two types of objects met. A good example to test the algorithm should have well-understood places where different types met. Thus, we can examine whether the algorithm gives good answers in those places. Therefore, we have

	MSE	ASC1	ASC4	SC(1-3)	SC(1-4)	MC
Number of Triangles	503813△	500789△	136852△	507986△	381030△	503594△
CPU Time (seconds)	66.08s	184.97s	53.57s	33.00s	35.09s	32.50s

Table 3.2: Comparsion of Multi-body Surface Extraction algorithm with Adaptive Skeleton Climbing (ASC, $N = 1, 4$), Skeleton Climbing (SC, without or with step 4 [33]) and Marching Cube (MC), for extraction of a single isosurface from "cthead" data in resoultion $256 \times 256 \times 113$.

created two volumetric data based on the mathematical equations defined as following:

$$\chi_{(p,q,r)}(x,y,z) = \begin{cases} 1 & \text{if } ((x-p)^2 + (y-q)^2 + (z-r)^2) < R^2 \\ 0 & \text{otherwise} \end{cases}$$

$$\begin{aligned} K_1(x,y,z) &= \max\left\{\chi_{(1,0,0)}(x,y,z), 2\chi_{(-1,0,0)}(x,y,z), 3\chi_{(0,1,0)}(x,y,z)\right\} \\ K_2(x,y,z) &= \max\left\{K_1, 4\chi_{(0,-1,0)}(x,y,z), 5\chi_{(0,0,1)}(x,y,z), 6\chi_{(0,0,-1)}(x,y,z)\right\} \end{aligned}$$

and we use $R = 1$ and K_1 to create the first data – "4types", and $R = 1$ and K_2 to create the second data – "7types". The resolutions of the data are both $128 \times 128 \times 128$. **Fig. 3.19** and **Fig. 3.20** display all the isosurfaces with different colors, and **Fig. 3.21** displays the close-up views of the places where several surfaces met.

Volumetric data "head2" is a medical CT-scanned data. **Fig. 3.22** displays the multi-body surface extracted and there are surfaces of two types of objects – muscle and bone, as shown in **Fig. 3.23** and **Fig. 3.24**. Image segmentation and registration are first applied to the MRI-scanned data, "frog", "orange" and "tomato", to identify the different types of objects. Different gray levels are then assigned to the objects, and the multi-body surface extraction algorithm is used to generate the isosurfaces. **Fig. 3.25**, **Fig. 3.26** and **Fig. 3.27**, display the multi-body surfaces. Though the surfaces are no smooth due to the complex structure of the data, the surfaces of different objects can be easily identified.

Moreover, since the algorithm is a generalization of SC algorithm, by setting number of types to be only two, the algorithm works just as single surface extraction algorithm. **Fig. 3.28** shows the single isosurface extracted from another medical CT-scanned data "cthead". We also compare the algorithm with existing algorithms, in terms of the triangle counts and CPU time, in **Table 3.2**.

If the techniques of ASC are not applied to the binary blocks, the number of triangles extracted are enormous (in the order of 10^6). **Table 3.3** shows the quantitative result after incorporating with ASC algorithm, with block size $N = 1, 2, 4, 8$. The visual comparisons of the effects of using different block sizes to various data are shown in **Fig. 3.18**, **Fig. 3.19**, **Fig. 3.20**, **Fig. 3.23**, **Fig. 3.24** and **Fig. 3.28**. There are up to 80% reduction in the triangle counts, depending on the geometric complexity of the true isosurfaces. In general, as the block size N increases, both the triangle counts (**Fig. 3.16**) and CPU times (**Fig. 3.17**) decrease. However, in some cases, the increasing of N may slightly increase the triangle counts. It is because, the increasing of the block size N, will also increase the percentage of the non-binary blocks. Recall that in the non-binary blocks, triangles are extracted from the smallest sized cubes. Therefore, less adaptive and more triangles are generated in those blocks and the overall triangle counts may even greater than the cases using smaller block size. Moreover, the triangle counts still actually depend on the geometry of the actual isosurfaces, and complex isosurfaces requires sufficient triangles to represent it. In most cases, the optimal block size (in terms of both triangle counts and CPU time) is 4. Depending on the geometric complexity of the isosurfaces and number of types of objects, this optimal value may vary.

Volumetric Data	MASC1 $N = 1$	MASC2 $N = 2$	MASC4 $N = 4$	MASC8 $N = 8$
spheres ($128 \times 128 \times 128$)	228156△ 50.28s 0.00%	62862△ 22.83s 0.00%	24000△ 15.65s 0.00%	122280△ 17.65s 0.00%
4types ($128 \times 128 \times 128$)	185006△ 40.37s 0.05%	54242△ 19.76s 0.21%	28613△ 13.94s 0.84%	34607△ 15.73s 3.47%
7types ($128 \times 128 \times 128$)	237846△ 47.92s 0.16%	87744△ 25.33s 0.76%	73522△ 18.57s 3.10%	106902△ 20.99s 12.48%
head2 ($256 \times 256 \times 98$)	683718△ 149.33s 0.023%	229163△ 61.76s 0.27%	242101△ 44.97s 1.73%	351874△ 48.88s 5.58%
cthead ($256 \times 256 \times 113$)	500752△ 125.09s 0.00%	170116△ 48.27s 0.00%	137672△ 33.55s 0.00%	152724△ 35.46s 0.00%
tomato ($128 \times 128 \times 64$)	357300△ 52.76s 1.24%	209805△ 27.90s 4.71%	240627△ 20.29s 17.02%	325424△ 22.65s 38.13%
orange ($128 \times 128 \times 64$)	332234△ 48.34s 1.88%	267783△ 27.14s 8.00%	319256△ 20.07s 20.14%	330350△ 21.95s 34.47%
frog ($128 \times 128 \times 136$)	456765△ 73.34s 0.94%	303026△ 35.70s 3.33%	352333△ 26.62s 9.29%	414816△ 26.28s 19.51%

Table 3.3: Quantitative results after incorporating with ASC, with block size $N = 1, 2, 4, 8$. (First row is the number of triangles extracted. Second row is the CPU time in seconds. Third row is the percentage of non-binary blocks.)

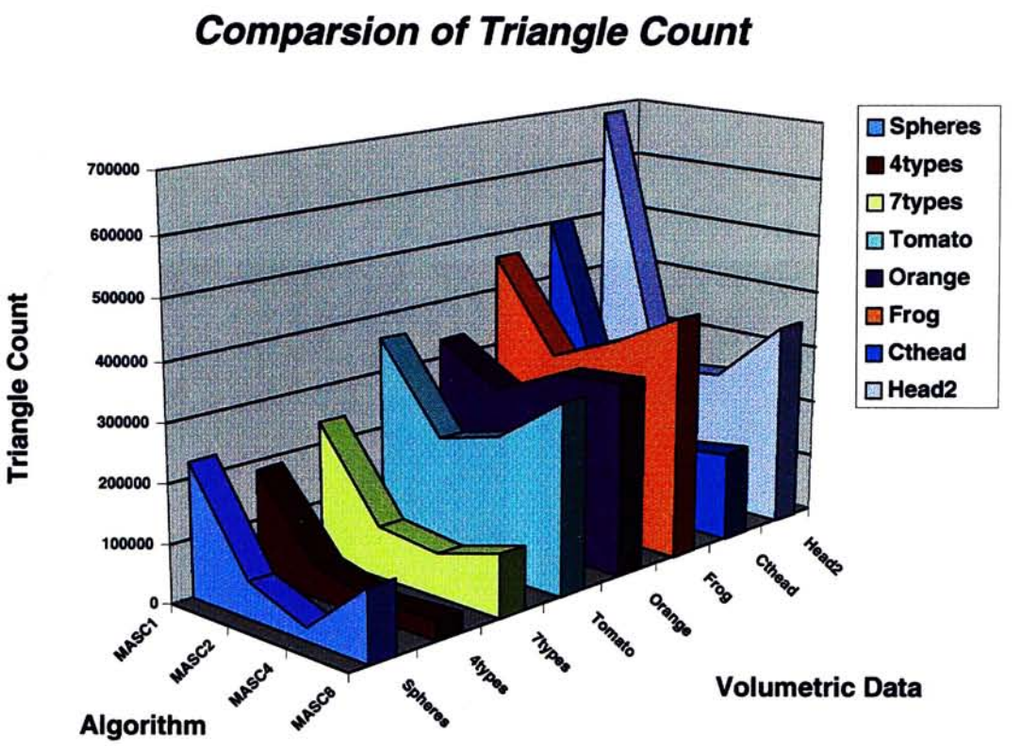


Figure 3.16: Comparison of triangle count with different block size.

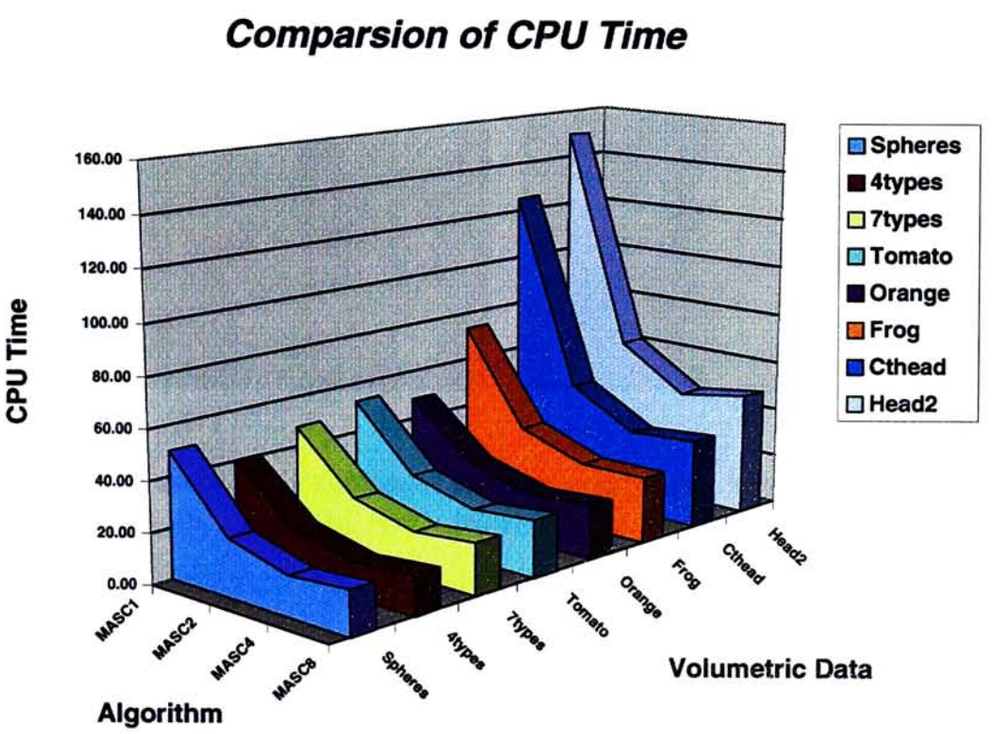
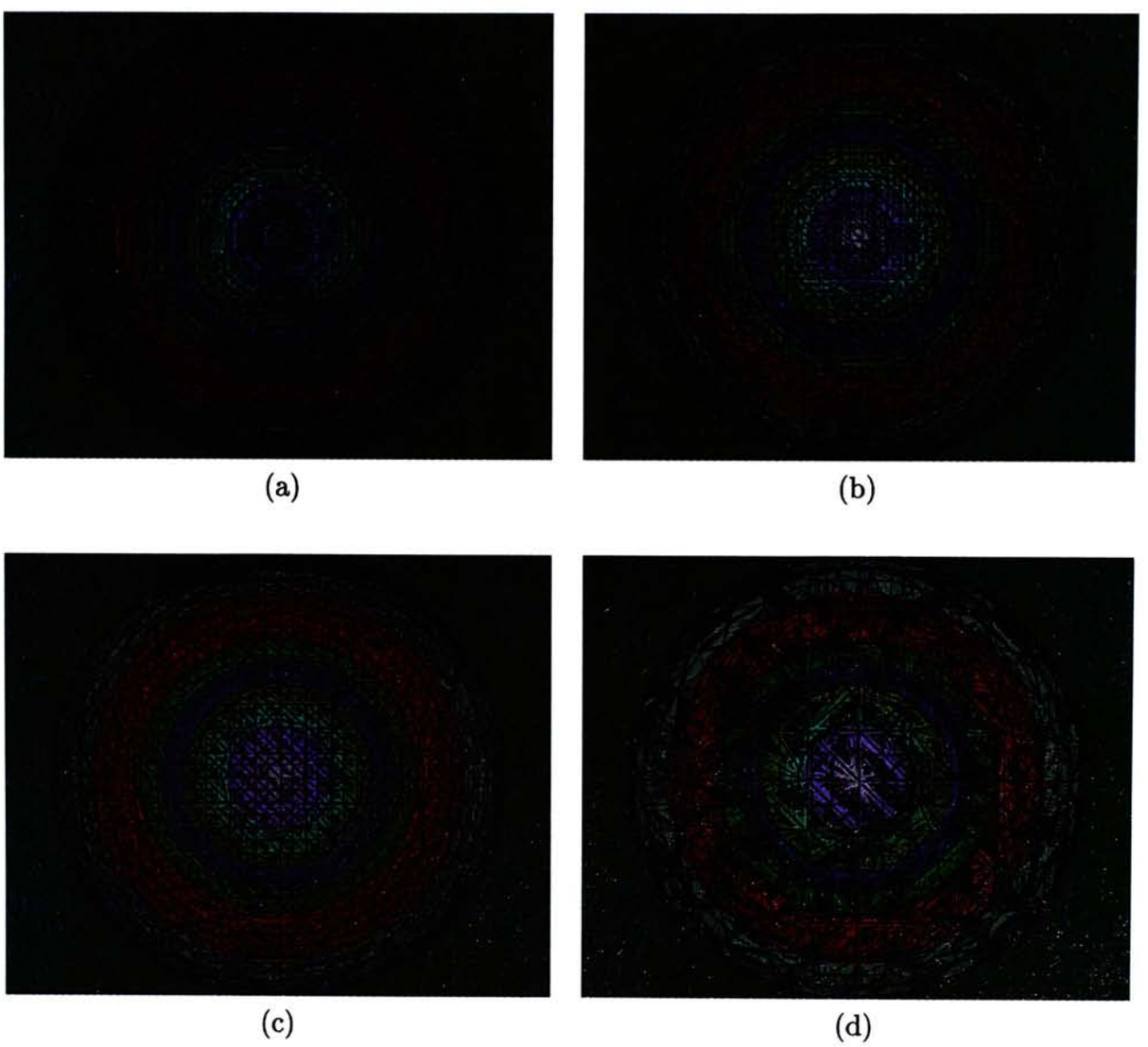


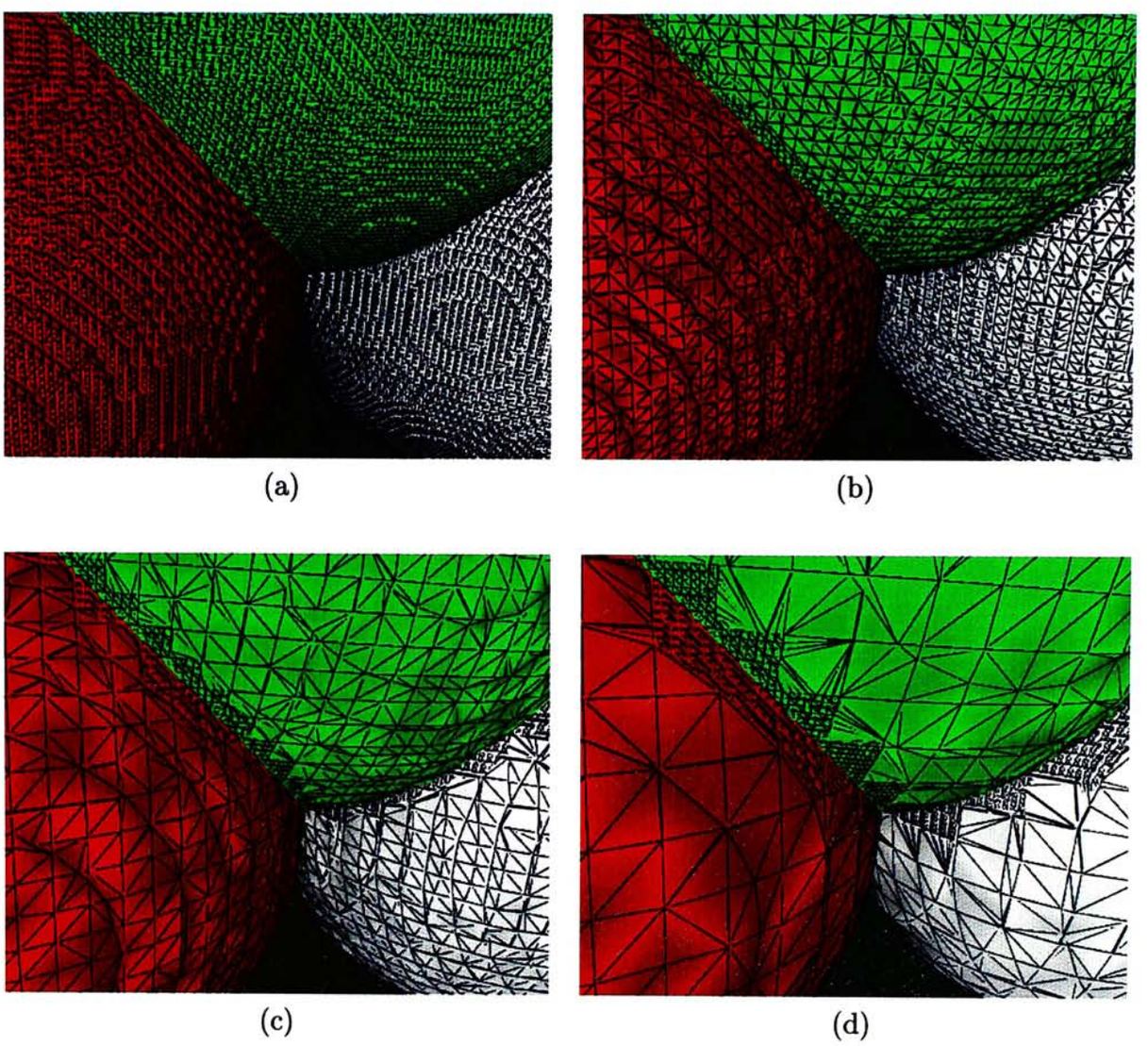
Figure 3.17: Comparison of CPU time with different block size.



(a) (b)

(c) (d)

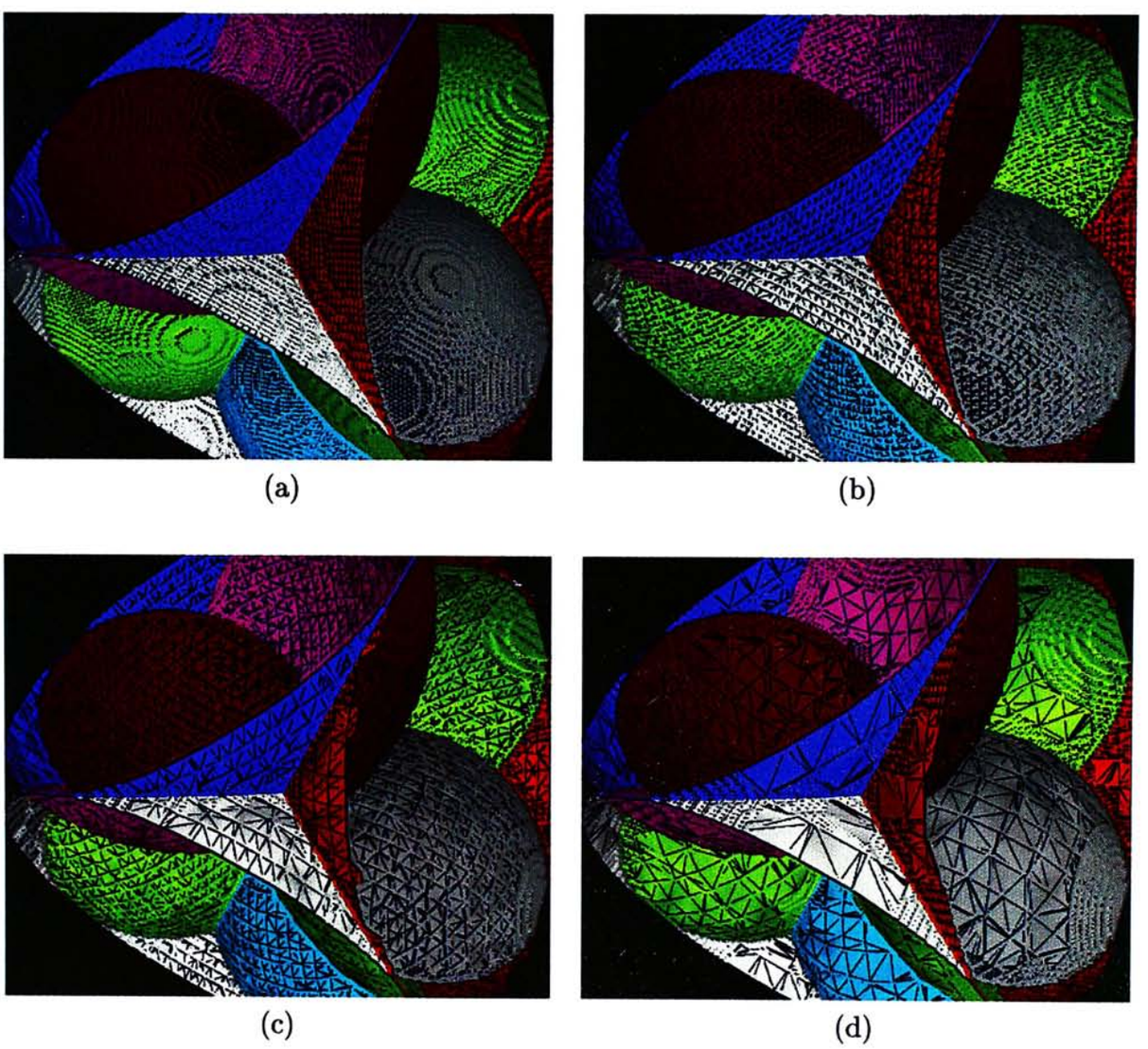
Figure 3.18: The multi-body surface of the "spheres" data. Visual comparison of the effects of block size, (a) N=1, (b) N=2, (c) N=4, (d) N=8.



(a) (b)

(c) (d)

Figure 3.19: The multi-body surface of the "4types" data. Visual comparison of the effects of block size, (a) N=1, (b) N=2, (c) N=4, (d) N=8.



(a) (b)

(c) (d)

Figure 3.20: The multi-body surface of the "7types" data. Visual comparison of the effects of block size, (a) N=1, (b) N=2, (c) N=4, (d) N=8.

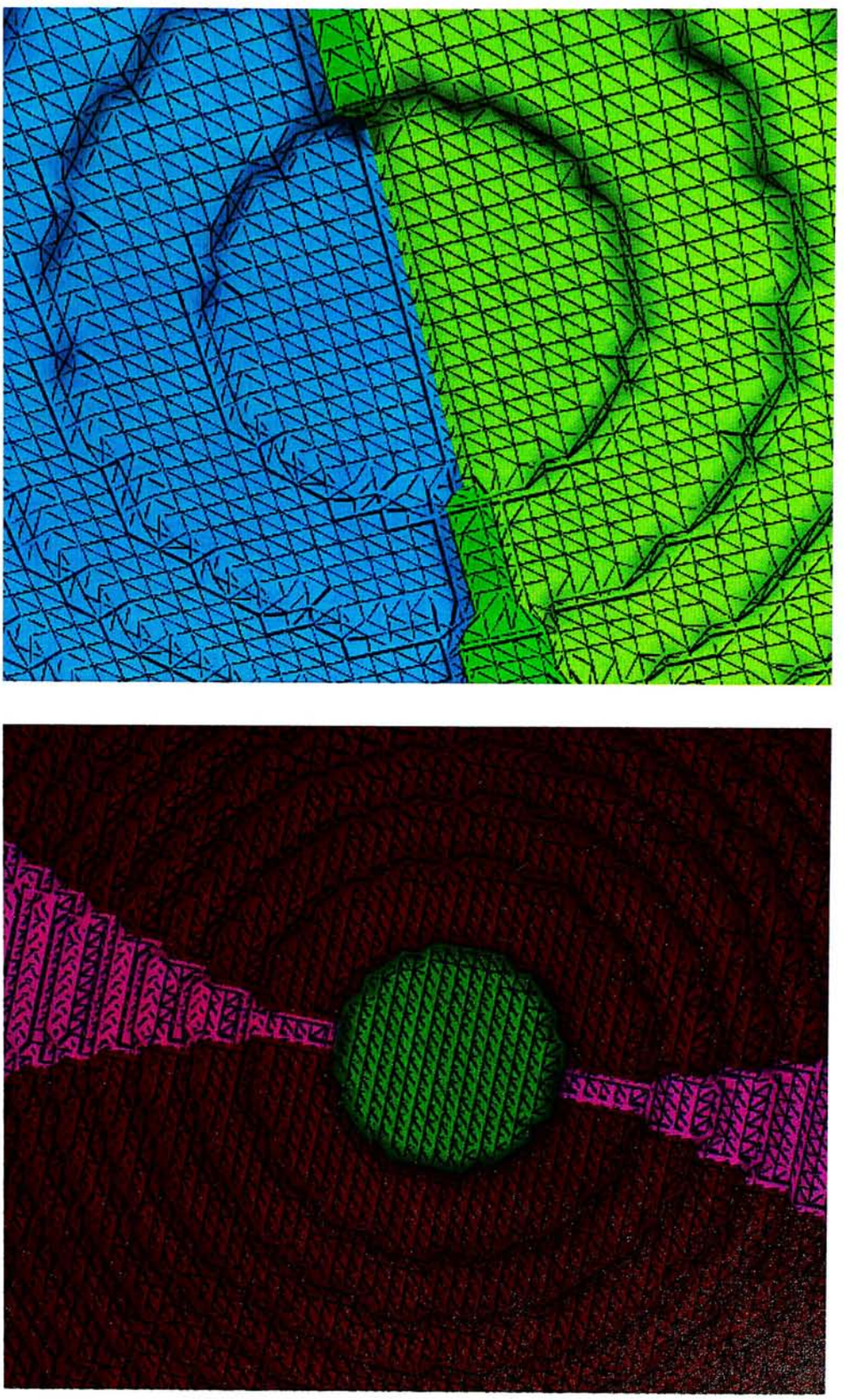


Figure 3.21: Close-up views of the places where multiple types met, zooming of the "4types" and "7types" data.

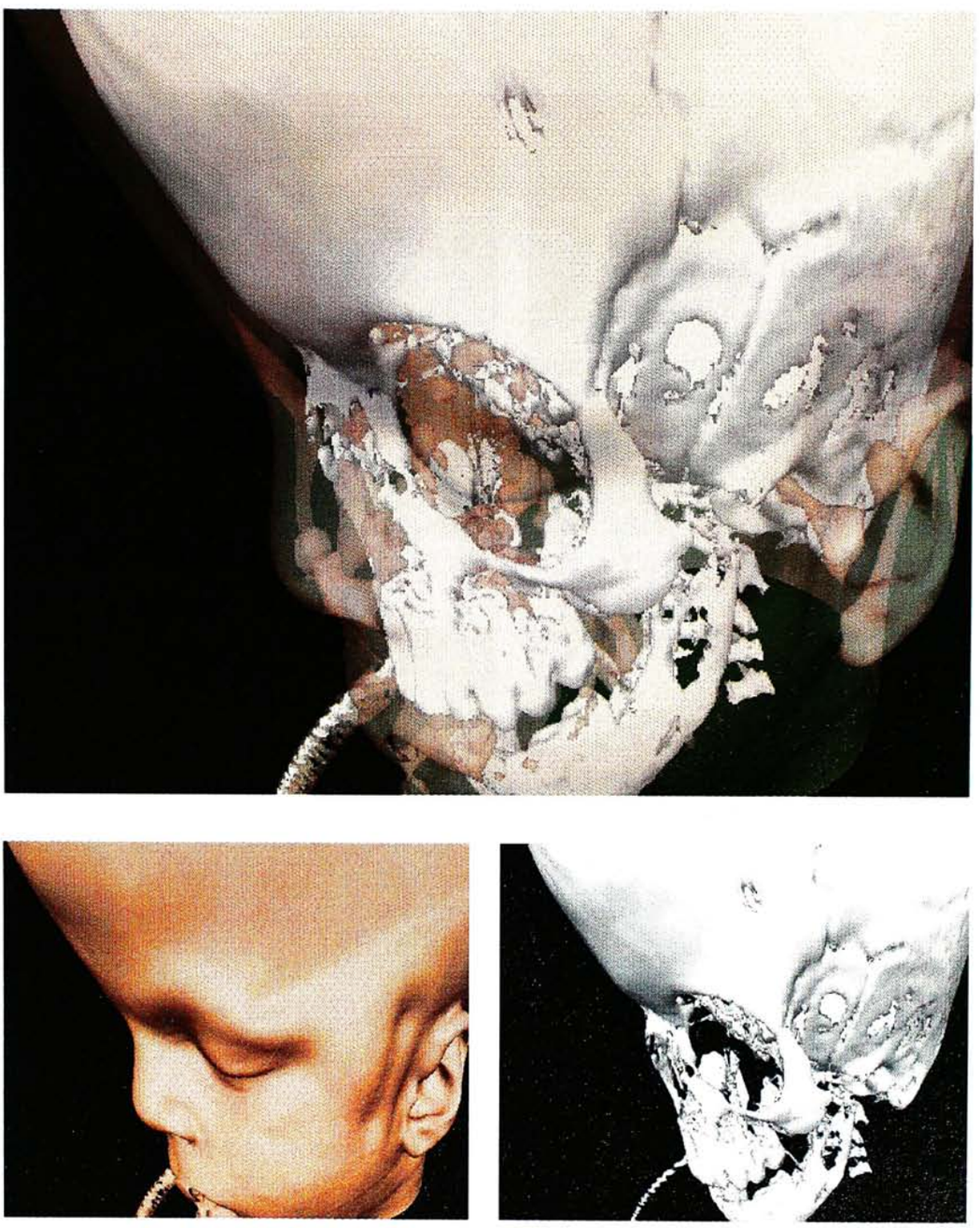


Figure 3.22: The multi-body surface of the "head2" data.

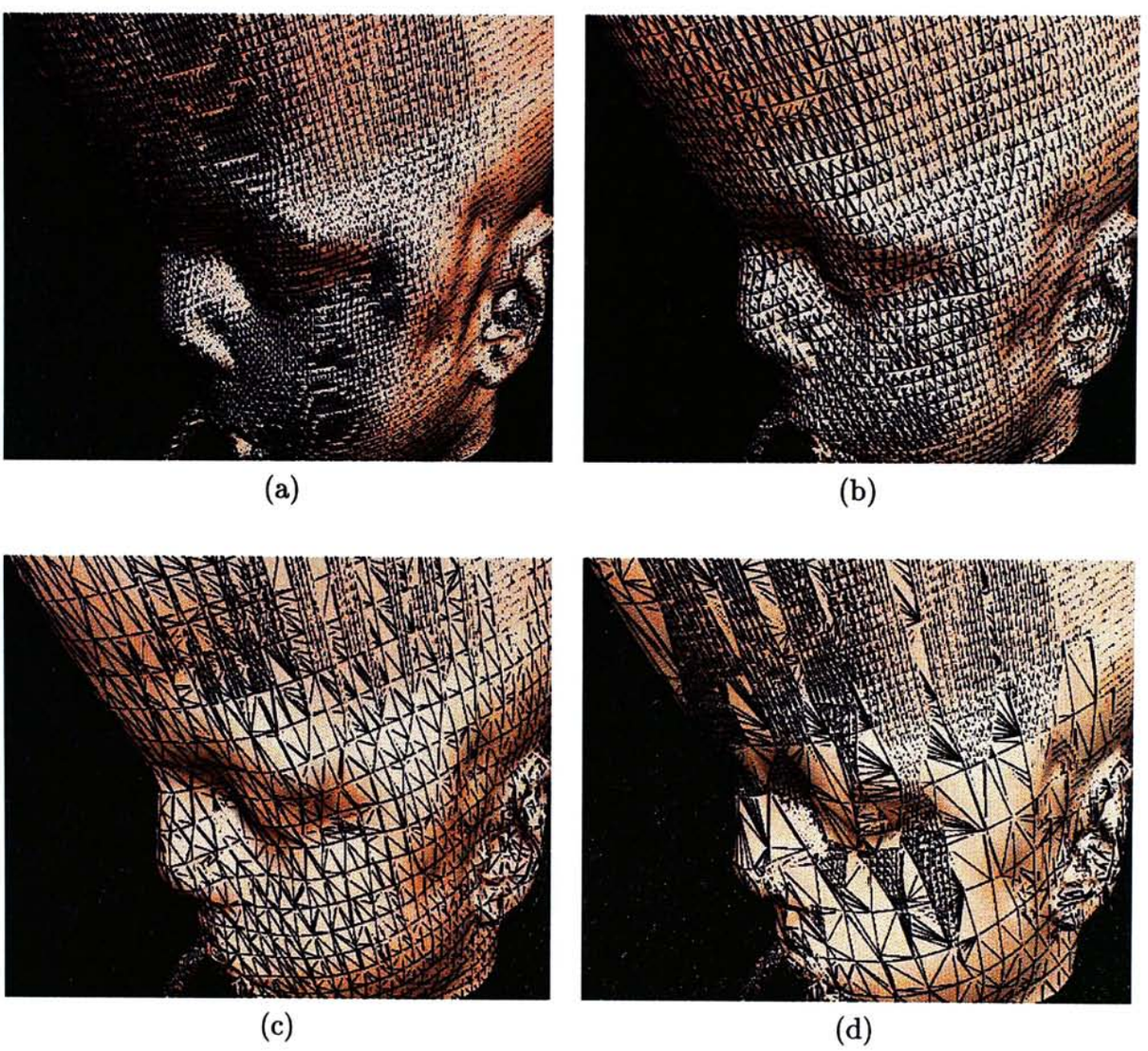
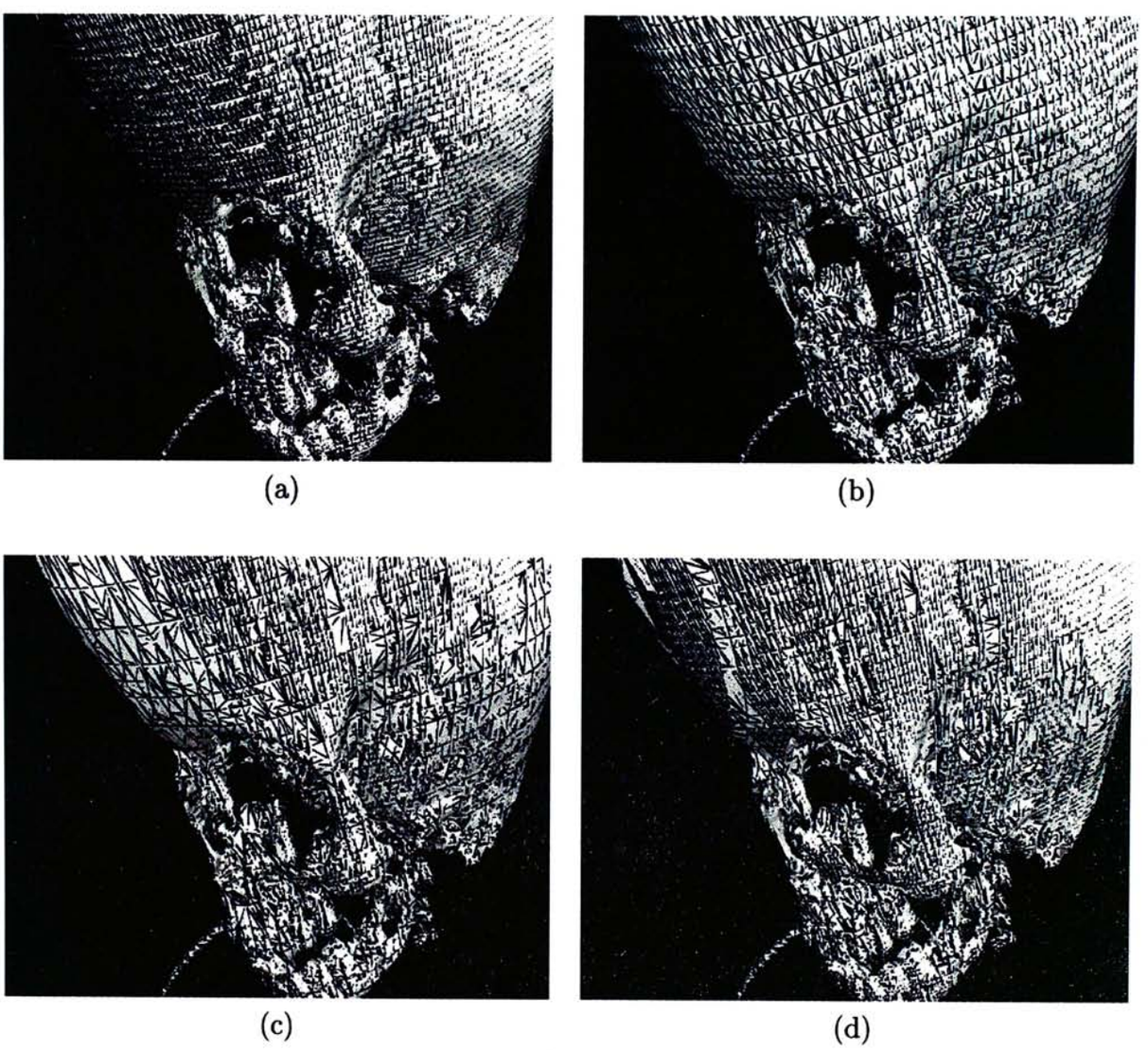


Figure 3.23: Visual comparison of the effects of block size, for "muscle" surface of the "head2" data. (a) N=1, (b) N=2, (c) N=4, (d) N=8.



(a) (b)

(c) (d)

Figure 3.24: Visual comparison of the effects of block size, for "bone" surface of the "head2" data. (a) N=1, (b) N=2, (c) N=4, (d) N=8.

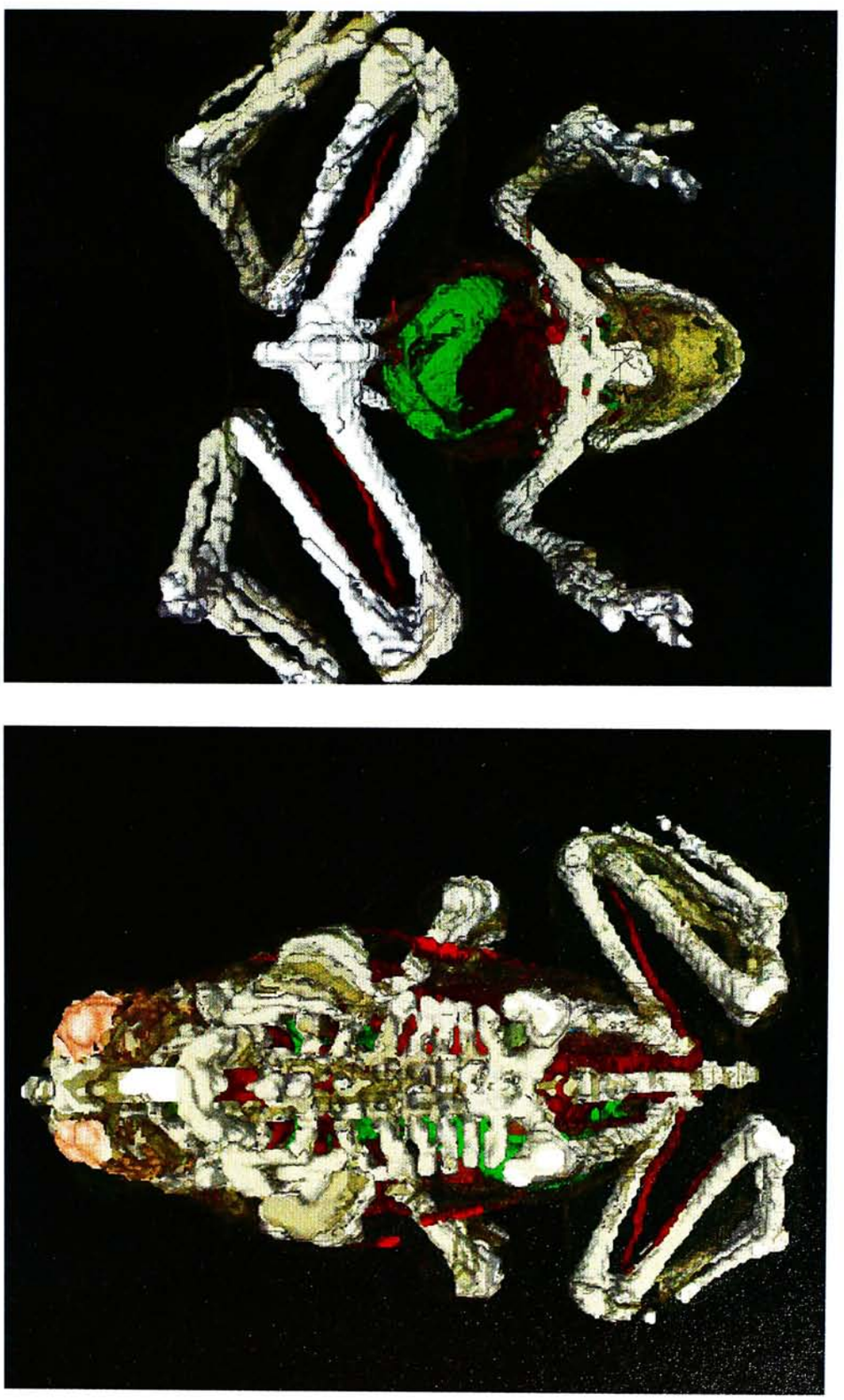


Figure 3.25: The multi-body surface of the "frog" data.

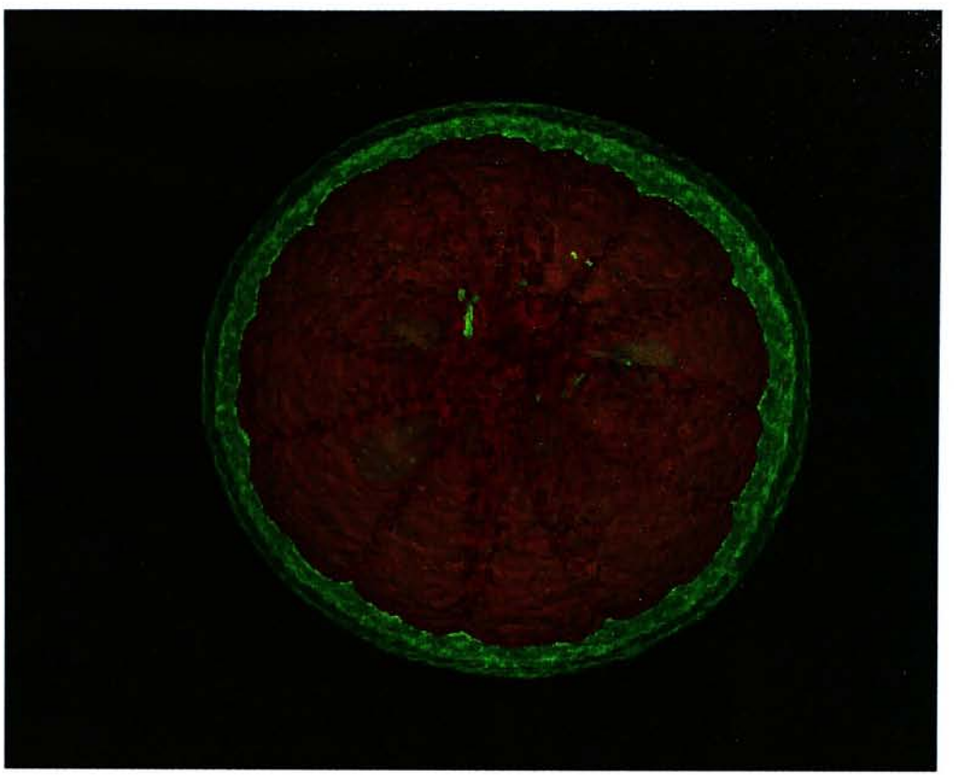


Figure 3.26: The multi-body surface of the "orange" data.

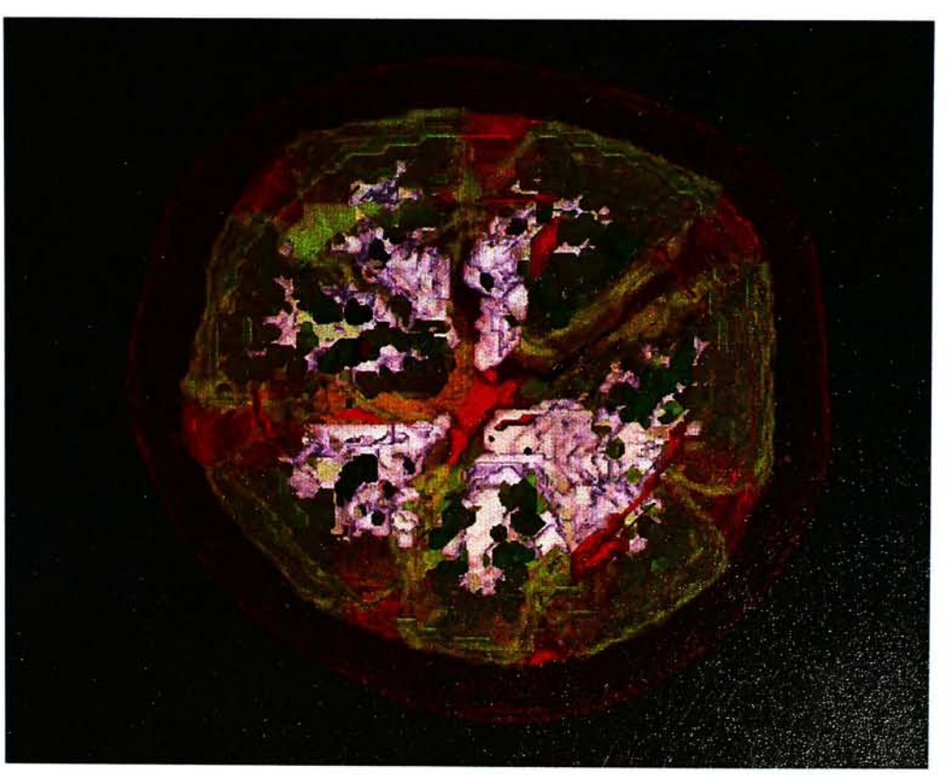
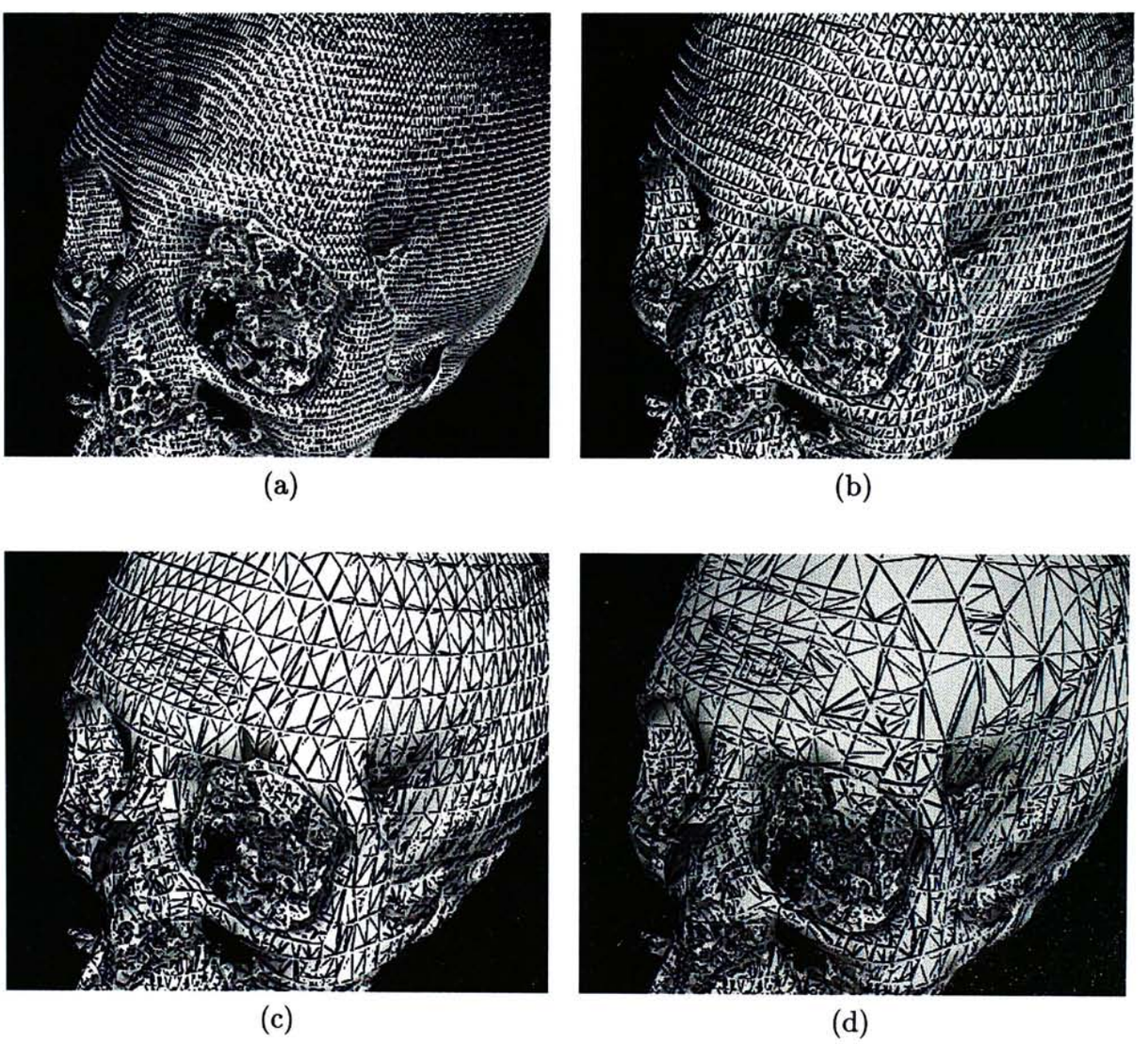


Figure 3.27: The multi-body surface of the "tomato" data.



(a) (b)

(c) (d)

Figure 3.28: Visual comparison of the effects of block size, for the "cthead" data. (a) N=1, (b) N=2, (c) N=4, (d) N=8.

Part II

Haptic Rendering of Volumetric Data

Chapter 4

Introduction to Haptics

Haptics refers to manual interactions with environments, such as exploration for extraction of information about the environment or manipulation for modifying the environments. These interactions may be accomplished by human or machine hands and the environment can be real or virtual. Also, the interactions may or may not be accompanied by other sensory modalities such as vision and audition. Most of the virtual environments (VEs) built to date contain visual displays, primitive haptic devices such as trackers or gloves to monitor hand position, and spatialized sound displays. To realize the full promise of VEs, haptic displays with force and/or tactile feedback are essential. It is quite likely that much greater immersion in a VE can be achieved by the synchronous operation of even a simple haptic interface with a visual and auditory display, than by large improvements in, say, the fidelity of the visual display alone.

Fig. 4.1 shows the subsystems of a haptic VE (*Human sensorimotor* loop and *Machine sensorimotor loop*), and information flow underlying interactions between human users and force-reflecting haptic interfaces.

Human Sensorimotor Loop – When a human user touches a real or virtual object, force are imposed on the skin. The associated sensory information is conveyed to the brain and leads to perception. The motor commands issued by the brain activate the muscles and result in hand and arm motion.

Machine Sensorimotor Loop – When the human user manipulates the end effector of the haptic interface device, the position sensors on the device convey its tip position to the computer. The models of objects in the computer calculate in real-time the torque commands to the actuators on the haptic interface, so that appropriate reaction forces are applied on the user, leading to tactile perception of virtual objects.

The sections below are organized as follow. In **Section 4.1**, we will clarify the ter-

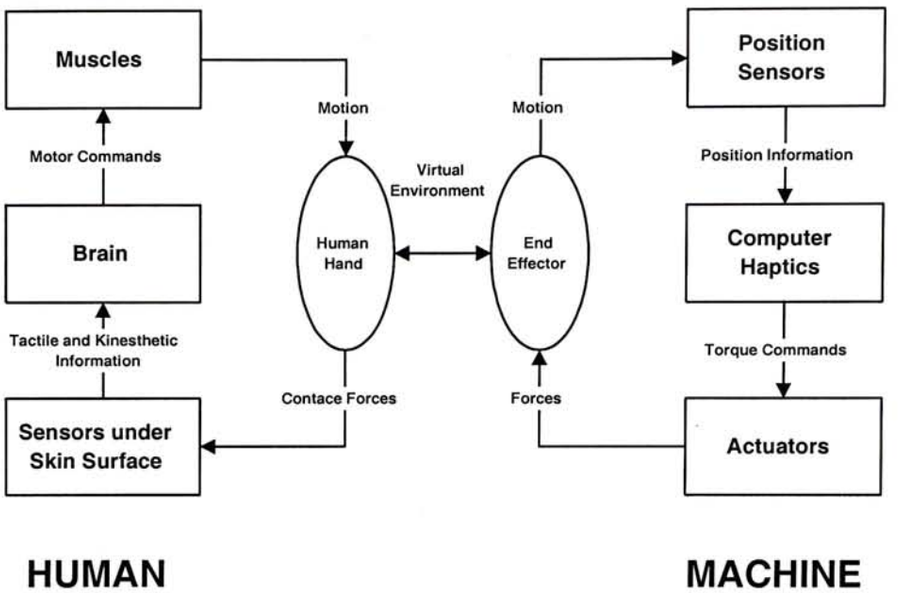


Figure 4.1: Haptic interaction between humans and machines.

minology concerning both human and machine aspects of this rapidly developing field. Then we will discuss the process of haptic rendering in **Section 4.2**. In **Section 4.3**, we give primary classifications of haptic interfaces and briefly describe the hardward device used in this research – the PHANToM™ haptic interface. Finally, we state our research goals at **Section 4.4**.

4.1 Terminology

According to Webster dictionary, the definition of *haptic* is "relating to or based on the sense of touch". As an active field of research, haptics is relatively new. It is better to define the terminology of this new field, before we go into the detailed discussion.

There are three major components of haptics in VEs, namely *human haptics*, *haptic interfaces* and *computer haptics*.

Human Haptics – There are two classes of touch sensations of human in contact with an object: (1) *tactile information*, referring to the sense arising from the skin in contact with the object; (2) *kinesthetic* (or *proprioceptive*) information, referring to the sense of position and motion of limbs along with the associated forces. In general, net forces of contact are sensed by both the systems, but the spatiotemporal force variations within the contact region are conveyed by the tactile system alone. Consequently, the fine shape, texture, and compliance of the object within the contact region are sensed

by tactile sensors in the skin. The coarser properties of objects, such as spring-like compliance that required hand or arm motion for exploration, are conveyed by the kinesthetic system.

Haptic Interfaces – Haptic interface is a link between the human operator and a virtual environment, includes a haptic device and any software required to ensure stable interaction. In interacting with VEs using a haptic interface, the human user conveys desired motor actions by physically manipulating the interface, which in turn, displays tactual sensory information to the user by appropriately stimulating his or her tactile and kinesthetic sensory systems. At present force display devices that can match at least some of the capabilities of the human haptic system are available.

Computer Haptics – Computer haptics is the discipline concerned with generating and rendering haptic stimuli to the human user, just as computer graphics deals with generating and rendering visual images. As computers become more powerful and affordable, and sophisticated software tools and techniques are increasingly available to the human user, the need for more effective interactions between humans and computers becomes urgent. Therefore, we expect that similar rapid progress in computer graphics to occur in computer haptics, as indicated by recent papers on PHANToM™-based applications [38, 39, 40, 41].

4.2 Haptic Rendering Process

Haptic rendering refer to the computational methods used to determine the forces resulted when we interact with virtual objects. Just as in computer graphics, the representation of 3D objects can be either surface-based or volume-based for the purposes of computer haptics. While the surface models are based on parametric or polygonal representations, volumetric models are made of voxels. An alternative way of distinguishing the existing haptic rendering techniques is based on the type of haptic interaction: *point-based* or *ray-based* [40].

For the point-based haptic interactions, the end-point location of the physical haptic interface as sensed by the haptic hardware device is simply modeled as a point. For the ray-based haptic interactions, the generic probe of the haptic device is modeled as a finite ray whose orientation is taken into account, and collision are checked between the ray and the objects. Both techniques have their pros and cons. For example, it is computationally less expensive to render 3D objects using point-based technique. Hence higher haptic servo rates can be achieved. On the other hand, the ray-based

haptic interaction technique handles side collisions and can provide additional haptic cues for conveying to the user the shape of objects. The ray-based algorithms are still in infancy. Hardware devices are difficult to design and haptic servo rate achieved is low. Many hardware devices available nowadays permit users to feel and control the forces arising from point interactions with virtual object. The point interaction paradigm greatly simplifies both device and algorithm development while permitting bandwidth and force fidelity that enable a surprisingly rich range of interaction.

4.2.1 The Overall Process

Creating a virtual haptic environment involves several steps. In general, objects are defined in their own coordinate system (object-space), and then transformed into world-space coordinates. Once the location of all objects, including the the position of the end-effector of the haptic interface for a point interaction (haptic interface point, HIP) is known, the steps for simulation can be summarized as follows:

1. Use collision detection algorithms to determine if the HIP has collided with any objects.
2. For all objects the HIP is touching, compute the constraint force from the appropriate force profile.
3. Sum all forces at the HIP.
4. Transform resultant force to torques and send them to the haptic device.

Having been transformed to the world coordinates, collision detection is performed to determine if any of the objects have intersected the HIP. Numerous effective collision detection algorithms have been developed for computer animation and interactive graphics displays, and techniques used in virtual environments can typically be applied to haptic displays since haptic simulations are essentially virtual environments. The challenge is to perform the necessary computations in real time. However, it is not sufficient to know that a collision occurred, but the location and amount of inter-penetration must be known. This need for both speed and accuracy presents a trade-off with limited computational power. If the HIP has collided with an object, a displacement vector is then defined, providing information about position and distance (the magnitude of displacement) of the HIP with respect to the object. For example, when intersecting a plane, the penetration depth in the direction of the surface normal gives the displacement. The displacement can be viewed as the vector from the HIP

to a goal point, thus making the displacement a measure of error to minimize. In this example, this goal point is the closest point to the HIP on the plane.

The force profile of the object then uses the position, velocity, and displacement to determine the magnitude and direction of the resultant force vector, as discussed below. Finally, in order to feel the generated force with a haptic interface, the force must be transformed into motor torques, which are defined in the joint space (object space) of the haptic device. The transformation is defined by the kinematics of the haptics interface.

4.2.2 Force Profile

In order to display an object with a haptic interface, the object must have a force, or impedance, representation to describe how it will "feel". This representation is its force profile, which is a function of displacement, position, velocity, or any combination of these position-related variables. Moreover, force profiles are also constrained by the system in which they are defined. The system generally consists of the haptic device, the computer hardware for computations within the simulated environment, the communication between devices, and even the user. The reaction force ($\mathbf{F}$) is usually calculated by a spring model (Hooke's Law) and expressed as

$$\mathbf{F} = K\mathbf{d} \tag{4.1}$$

where where $\mathbf{d}$ is the displacement into the surface. This is also known as proportional control to minimize the displacement into the surface. For rigid objects, the value of K is set as high as possible, limited by the contact instabilities of the haptic device. Studies show that addition of a damping term into interaction dynamics improves the stability of the system and the haptic perception of 'rigidity', as shown in **Fig. 4.2**. This is consider to be a proportional differential (PD) controller, and is the most commonly implementation in the literature [1, 25, 36, 50]. The force profile is expressed as

$$\mathbf{F} = K\mathbf{d} - B\mathbf{v}_\mathbf{d} \tag{4.2}$$

where $\mathbf{v}_\mathbf{d}$ is the velocity along $\mathbf{d}$. To create a wall that feels stiffer, an integral term can be added (PID control), but special considerations must be made to assure that $\mathbf{F} = 0$ outside the wall [41]. Other forces such as those that arise from surface friction and roughness also need to be displayed to improve the realism of haptic interactions.

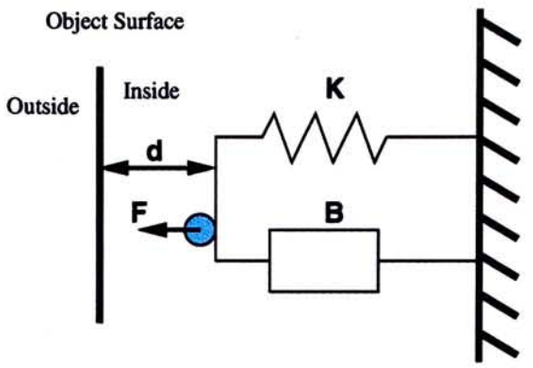


Figure 4.2: Spring-damper model to compute the force profile.

Since the HIP can collide with multiple objects simultaneously, the force vector for each interaction must be computed, and then added together.

$$\mathbf{F}_{total} = \sum_{i=1}^{M} \mathbf{F}_i \tag{4.3}$$

where $\mathbf{F_i}$ is the force exerted by the ith object, and M is the total number objects being checked. A weighted sum might be necessary to scale the force, but the determination of the weights is very dependent on the desired effect.

4.2.3 Decoupling Processes

Human's tactile sensory system is much more sensitive than the visual sensory system, where the spatiotemporal force vibrations of up to 1000 Hz are resolvable. To incorporate haptics into a multi-sensory virtual environment, the haptic and application/graphic processes are often separated to run in a client-server architecture [1, 25, 36]. Decoupling of the processes is important since the haptic rendering process must run at a very high rate and degrade gracefully and safely as the complexity of the environment is increased. Moreover, this will help ensure that if there is a delay in one process the problem will not affect overall performance and safety of the system.

The typical system architecture of a haptic simulation is shown in **Fig. 4.3**. The bulk of haptic rendering effort is placed on the haptic server, thus freeing the client to perform the tasks required by the user's application. The haptic server receives high level commands from the client, tracks the position of the haptic device, updates the position of the haptic interface point, and sends control commands to the haptic device. The haptic server and client application run in parallel as processes of either a single computer, or different computers communicated via TCP/IP packets.

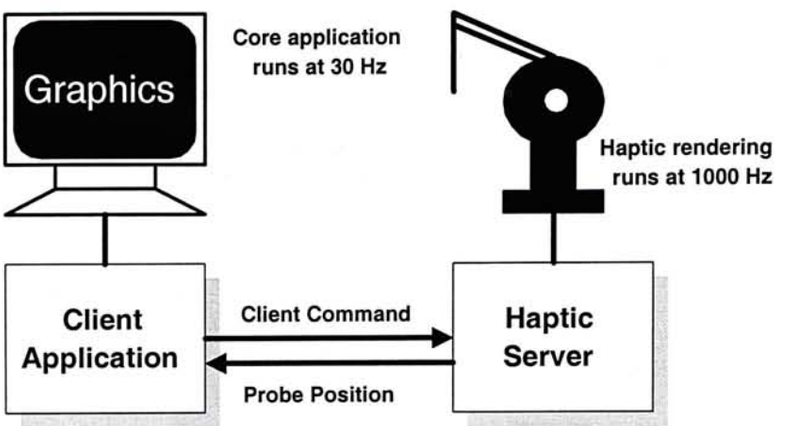


Figure 4.3: System architecture of haptic simulations.

4.3 The PHANToM™ Haptic Interface

Mechanical devices that allow haptic interaction with remote and virtual objects have been around for decades. Early remote manipulation systems were used for handling hazardous substances, as far back as the 1940s, when the danger of working with nuclear materials necessitated developing remote manipulation devices. Today's surge of haptic research and commercial activity grew from the early efforts of designers who built the "master" input devices needed to control remote manipulators. In the 1960s, Knoll at Bell Labs was perhaps the first to demonstrate touching virtual shapers with a computer-controlled haptic interface [38]. Since then, numerous devices have been built for haptic interaction, based on the recognition that adding haptics to graphic images significantly improves human-computer interactions.

An important distinction among haptic interfaces is whether they are tactile displays or net force displays. The corresponding difference in interactions with VEs is whether the direct touch and fell of objects contacting the skin is simulated or the interactions are felt through a tool. An alternative distinction among haptic interfaces is whether the interface is *ground-based* or *body-based.* Force reflecting joysticks are examples of ground-based devices and exoskeletons represent body-based devices. Hybrid devices which combine both of these characteristics have also been built. Well-designed exoskeletal devices have the advantage that their kinematics and workspace coincide with those of the human. However, the design becomes complex because the unbalanced forces applied on, say the user's fingerpad, must eventually be grounded somewhere. In addition, the need for the user to carry the mass of the device can interfere with feeling objects in VE and can cause fatigue.

The Personal Haptic Interface Mechanism [26] (PHANToM™), **Fig. 4.4**, distributed

Figure 4.4: The PHANToM™ haptic interface.

by SensAble Technologies, Inc. has evolved as a result of a research at the MIT Artificial Intelligence Laboratory at 1993. The PHANToM™ is a convenient ground-based device which provides a force-reflecting interface between a human user and a computer. Users connect to the mechanism by simply inserting their index finger into a thimble. The PHANToM™ tracks the motion of the user's finger tip and can actively exert an external force on the finger, creating compelling illusions of interaction with solid physical objects. A stylus can be substituted for the thimble and users can feel the tip of the stylus touch virtual surfaces. By stressing design principals of low mass, low friction, low backlash, high stiffness and good backdrivability, the system have been developed is capable of presenting convincing sensations of contact, constrained motion, surface compliance, surface friction, texture and other mechanical attributes of virtual objects. **Table 4.1** shows the specification of PHANToM™ haptic device.

Nominal position resolution	0.03 cm
Workspace	13x18x25 cm
Backdrive friction	0.04 N
Maximum exertable force	8.5 N
Closed loop stiffness	3.5 N/mm
Inertia (apparent mass at tip)	< 75 g
Footprint	25x33 cm

Table 4.1: The specification of PHANToM™ haptic device.

4.4 Research Goals

The goals of this research is to derive methods of haptically rendering volumetric data, especially isosurfaces which may represent hard structures, such as bone in anatomical data. Haptic rendering methods which based on spring-damper model (**Eqn. 4.2**), are well defined and studied extensively [1, 25, 36, 50]. However, traditional approaches of haptic rendering of solid surfaces have been largely limited to primitives consisting only of geometric models. Therefore, to render a volumetric data one approach employs isosurface extraction methods, such as discussed in **Part I**, to obtain an explicit geometric structure. Although, it is useful to some extent, it lacks flexibility. Moreover, the number of geometric primitives produced is still a challenge for the haptic rendering algorithms in terms of speed as well as realism. The other approach is to direct haptically render the volumetric data just as the traditional visual volume rendering [16, 2, 14, 3]. The methodology of volume haptization [16, 2] allows haptic palpation and modification of volumetric data without preprocessing steps, but lacks the ability to simulate stiff walls (due principally to the lacks of specific surfaces), an important feature for proper rendering of hard anatomic structures such as bone.

We propose a direct haptic rendering method for isosurface in volume data using a point-based haptic feedback device, Our algorithm uses a virtual plane as an intermediate representation of the isosurface, and computes the point interaction force applied to the haptic interface based on this virtual plane. Using this approach, we are able to gain higher servo rate for complex volumetric data. Moreover, it makes maintenance of the stability of the simulation easier, and applicable to noisy data without preprocessing. We have developed our algorithm and tested with synthetic data and medical data, using the PHANToM™ haptic interface.

Chapter 5

Haptic Rendering of Geometric Models

The haptic rendering methods of geometric models can be classified into two groups. In *penalty based methods*, also known as vector field methods [26, 25], force proportional to the amount of penetration into a virtual volume are applied to the haptic device. However, this approach has a number of drawbacks. First of all, it breaks down for objects with complex polygonal mesh. Moreover, it also has problem when dealing with multiple objects and object with thin volume. Finally, force direction and magnitude computed may not be continuous across a sub-volume boundary. The drawbacks of penalty based methods led to solutions that keep a history of contact surfaces description of the objects, so that we always know the surfaces the haptic interface point has past through. The algorithms are grouped together under the name of *constraint based methods* [50, 36], they are well-designed for generating convincing interaction forces for objects modeled as rigid polyhedral. In this approach, a virtual position is defined which represents where the haptic interface point (HIP) will be located if the haptic interface could not penetrate the surface. The virtual haptic interface point (VHIP) is constrained by the object surface, which models real world object interaction of rigid bodies. The force computed is proportional to the displacement between the haptic interface point and the surface contact point. Moreover, A rich set of surface properties can be easily simulated by restricting or changing the motion of the VHIP.

In the following two sections, we will first discuss the penalty based methods and their drawbacks in **Section 5.1**, and then discuss the constraint based methods briefly in **Section 5.2**.

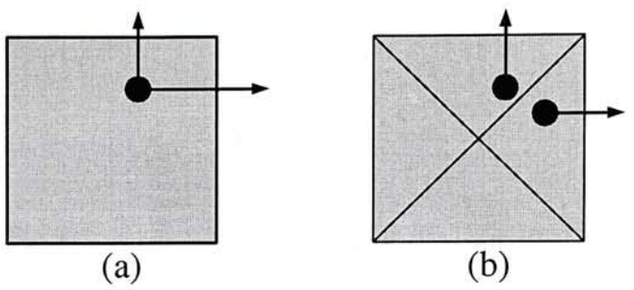


Figure 5.1: Force generation for a cube: (a) Without sub-volume division, (b) With sub-volume division.

5.1 Penalty Based Methods

5.1.1 Vector Fields for Solid Objects

In penalty based methods [26, 25], also known as vector field methods, force proportional to the amount of penetration into a virtual volume are applied to the haptic device. Vector field methods were derived from the fact that haptic interfaces do not have infinite bandwidth power, allowing the interface point to penetrate object surfaces. The feedback force is determined directly from this penetration depth. To avoid ambiguity of the force direction, there must be a direct one-to-one mapping from the position in the space within the object to a force vector direction. That is why these methods are also named vector field methods. To determine the direction of this force, the internal volume of an object is divided into sub-volumes [26], where each sub-volume is associated with a particular surface, or portion of the surface. The normal vector of that surface determines the force vector direction for the associated sub-volume. Therefore, once it has been determined which sub-volume the haptic interface point (HIP) is in, the force direction is given by the associated surface normal. The force magnitude is defined by the object force profile, such as **Eqn. 4.2**.

One of the simplest objects to render with this method is a plane. Here, the associated volume includes every point beneath the plane, and the force direction is the normal vector of the plane. A slightly more complicated object is a cube. **Fig. 5.1** shows a 2D slice of a cube, and displays the general problem and the sub-volume division. **Fig. 5.1(a)** shows that there are multiple solutions for the force direction at the point shown. A unique solution is found with the use of sub-volumes, as shown in **Fig. 5.1b**. For simple objects such as cubes, it is fairly simple to subdivide the internal volume by hand. Each cube facet essentially becomes the bottom of a pyramid, and all the six pyramids adjoin at the center of the cube. The force direction, in this case, is the normal vector of the bottom (cube facet) of the pyramid that contains the

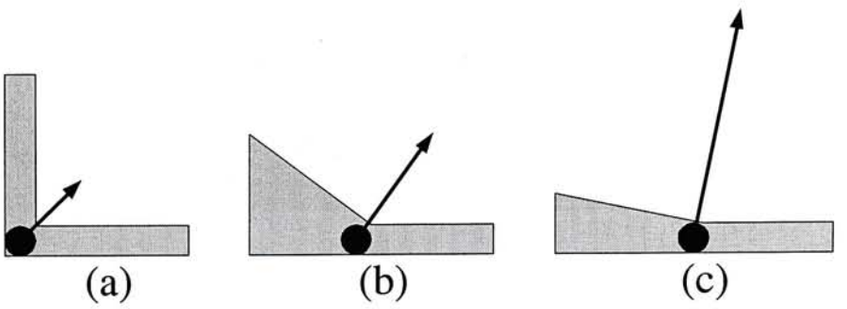


Figure 5.2: Force summation problem for multiple objects.

HIP. For another simple object such as sphere, every point within the sphere has a unique surface normal associated with it, which fits the definition of the vector field method. For complex polygonal mesh, it requires a complex algorithm to determin the sub-volume for each polygon surface. This approach has also been used for implicit surfaces such as volumetric data; however, determining the appropriate surface normal for every point beneath the surface is non-trivial.

5.1.2 Drawbacks of Penalty Based Methods

Penalty based methods have a number of drawbacks. First of all, it breaks down for objects with complex polygonal mesh, since it is not always clear how to subdivide a volume of complex object to give the most representative vector field. To haptically render objects of higher complexity, one solution is to combine (overlap) simple objects. In the regions where the simple objects intersect (corners and edges), the net reaction force is computed by vectorially adding contributions from each object's force field in a hope that it will generate the correct force sensations. However, this approach does not always produce accurate force vectors. For example, consider the intersection of two planes as shown in **Fig. 5.2**. When the angle between the two planes is approximately 90 degrees (**Fig. 5.2(a)**), the sum of the two displacement vectors results in an accurate representation. But as the angle approaches 180 degrees (**Fig. 5.2(c)**), the magnitude approaches twice that of one surface alone. This results in the surface stiffness is twice of the stiffness of each surface individually.

Another problem arises when the force direction and magnitude are not continuous across a sub-volume boundary. At the boundary transition shown in **Fig. 5.1(b)**, the direction changes by 90 degrees, but the magnitude is continuous because of the object symmetry. The discontinuity in direction gives the cube corner a "sharp" feels, as if plucking a string. However, when the sub-volumes have different shapes, such as **Fig. 5.3**, the discontinuity in both magnitude and direction change can generate

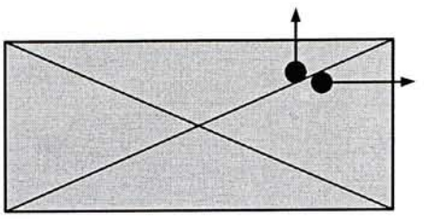


Figure 5.3: Force discontinuities can be encountered when traversing volume boundaries.

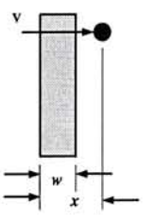


Figure 5.4: Representation of a thin object where the penetration distance x is greater than the object width w.

higher frequency, higher amplitude artifacts (instabilities).

There is also a minimum volume requirement, which thin or small objects may not meet. The vector field methods cannot accommodate for the possibility that the HIP can have a displacement greater than the width of the object, especially if the object is very compliant. **Fig. 5.4** shows an example of a thin object that has been penetrated from the left (in the direction of the velocity v). When the penetration distance x is greater than the object width w, the vector field methods cannot properly determine the force vector because the HIP is not within the volume of the object. The result is that the user feels passing through thin solid objects.

5.2 Constraint Based Methods

5.2.1 Virtual Haptic Interface Point

Constraint based methods [50, 36], deal with the problem of the penalty based methods, by keeping a history of contact surfaces description of the object. Due to the bandwidth limitation of haptic devices, we cannot prevent the HIP from penetrating the virtual object. It is however, we are free to define a virtual position which represents where the HIP will be located if the haptic interface could not penetrate the surface, as shown in **Fig. 5.5**. The virtual object represents this virtual haptic interface point (VHIP) is known as *god-object* in [50] or *proxy* in [36]. The VHIP is constrained by the object

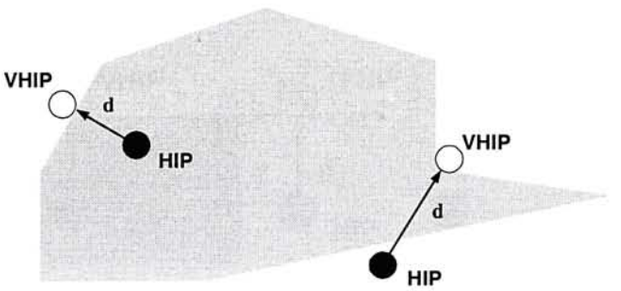


Figure 5.5: Constraint based methods - showing two possible interface points and their corresponding surface points.

surfaces, which models the interaction of rigid bodies in real world. Because VHIP remains on the surface of virtual object, the feedback force should never be ambiguous.

The main tasks of a constraint based method is to compute and keep the position of VHIP correctly. The resultant interaction force is then depending on the displacement from the HIP to VHIP. The basic algorithm steps are as follow:

1. Determine the location of the HIP.
2. Determine the VHIP on the surface that is closest to the HIP, given surface constraints.
3. Use the displacement from the HIP to VHIP to compute the resultant force.

5.2.2 The Constraints

The surface constraint methods use a virtual haptic interface point (VHIP) to describe a virtual object under the control of an algorithm in the haptic simulations. The surfaces of the virtual objects impede the motion of the VHIP, so it is considered as constraints. In free space, where there are no surface constraints, the VHIP is co-located with the HIP. As the HIP penetrates a surface, the VHIP remains on the surface. Its position is determined by a constraint minimization algorithm that locally minimizes the distance to HIP. From the force profile, the resultant force is a function of the displacement vector.

Consider the virtual object represented by polygonal facets, with a list of vertex coordinates and the vertices of each facet, the algorithm must determine which facets are constraining the motion of the VHIP. These facets are determined using the VHIP computed during the previous servo cycle, and the current HIP location as seen in

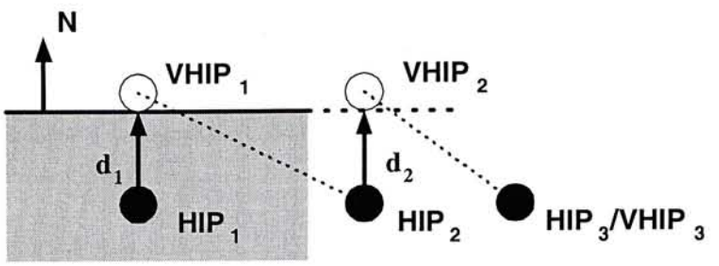


Figure 5.6: Surface constraint of a plane over 3 cycles.

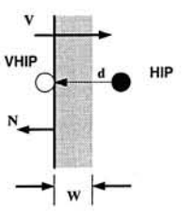


Figure 5.7: A thin object rendered by a constraint based method.

Fig. 5.6. If the algorithm decides that a particular facet is involved in the interaction, that facet becomes active. The active state is the condition where the VHIP is the front side of the rendered facet, and the HIP is on the other, and the line connecting the VHIP with the HIP intersects the facet within its boundaries.

Fig. 5.6 shows an example of a single constraint facet. At the first step, the $VHIP_1$ is computed as the closest point on the planar surface. At the second step, the HIP has crossed the boundaries, but the line connecting the current HIP_2 with the previous $VHIP_1$ crosses the surface within the boundaries. Therefore, the surface is still a constraint, and the $VHIP_2$ is the closest point to the plane, even though it is outside the boundaries. For the third step, the connecting line from HIP_3 to $VHIP_2$ does not cross the surface within the boundaries, so the $VHIP_3$ is unconstrained and becomes co-located with HIP_3.

The advantage of the constraint based methods for thin objects can be seen in **Fig. 5.7**, where the HIP has penetrated the other side of the object, in the direction of velocity $\mathbf{v}$. Since the VHIP has not yet crossed the edge boundary, the force direction is still toward the first facet as it would be if the surface had never been penetrated, i.e. the HIP does not "pop" through the object. Determining the active status for a single facet is trivial, but considering multiple facets at convex and concave intersections is more complicated.

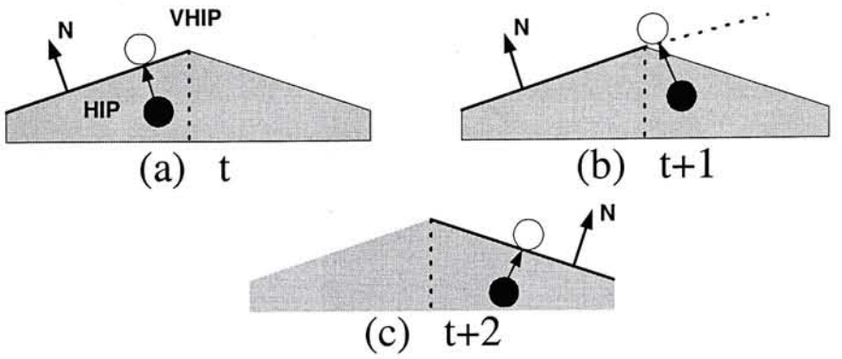


Figure 5.8: Transition between convex facets.

Convex Intersection

When two facets are connected at a convex intersection, there are two steps involved in the transition from one facet to the next, as shown in **Fig. 5.8((a)** to **(b)**, then **(b)** to **(c))**, which are similar to the steps for sliding off a single facet. With a convex intersection, only one facet can be considered active at any given time since a point cannot touch multiple convex planes simultaneously. **Fig. 5.8(a)** displays the HIP and VHIP before crossing the boundary, which is the plane bisecting the angle made by the intersecting facets. Similar to the example of a single facet, the first facet remains active at step t and step $t + 1$ and the new VHIP will remain on the plane of the first facet. At the next cycle, the first facet is no longer a constraint, and the VHIP falls to the second facet since this facet now meets the active condition, as shown in **Fig. 5.8(c)**. It is believed that *"the times and distances involved are small enough to cause only an imperceptible and transient distortion of shape,"* [50], although no experimental results have been published to verify this. The instantaneous changes in force direction and magnitude as the polygonal boundary is crossed, create the feeling of a distinct edge. The transition can be smoothed by the force shading method addressed in **Section 5.2.4**.

Concave Intersection

When probing a concavity, it is possible for multiple facets to be active at any given time. Consider feeling the inside of a box, when sliding along an edge, you are feeling resistance from two sides of the box. When touching a corner, you are feeling three. If two facets are felt simultaneously, the user is constrained to the line of the intersecting planes. With three constraints, only a point can be felt, which is the intersection of all three planes. Using a three DOF haptic interface, a maximum of three constraints can

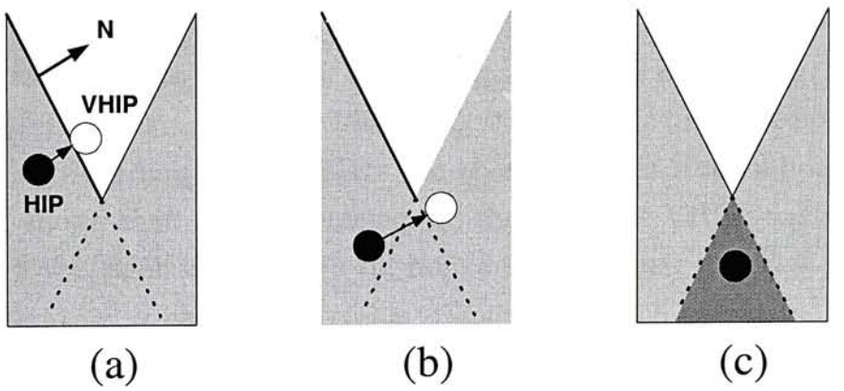


Figure 5.9: The problem of an acute concave intersection of surfaces.

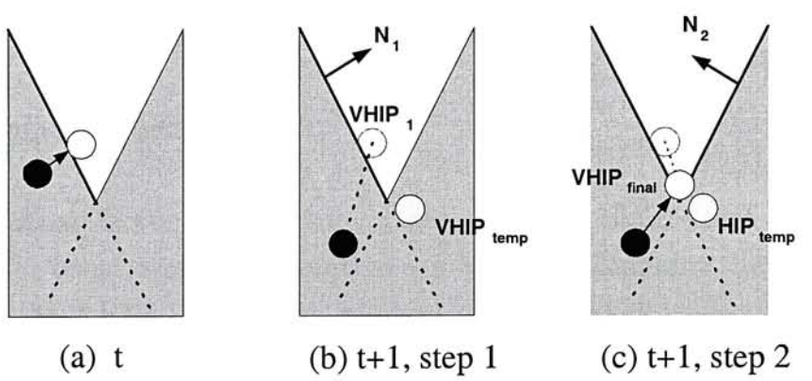


Figure 5.10: An iterative solution of the acute cancave intersection problem.

be represented. Additional constraints are redundant (feeling the inside of a faceted cone tip still results in feeling the point). As a save of computation time, it is no necessary to search for all the facets to find all the constraints. Since all constraining facets must intersect, once the first one is determined the search is limited to the adjacent facets.

Fig. 5.9 shows a problem occured at acute concave intersections of the object, when the user is sliding along the left facet toward the concavity. At the second servo cycle, **Fig. 5.9(b)**, the HIP has not crossed the boundary of the adjacent facet, and the line from the HIP to the previous VHIP still falls within the boundaries of the left facet (note that if the HIP crossed the boundary and enter the darker region in **Fig. 5.9(c)**, then both facets will be active and the VHIP is constrained to the intersection line). The new VHIP is then the closest point to the plane of the left facet. At the next cycle, the line between the HIP and the previous VHIP will not cross either facet, and the VHIP will become colocated with the HIP. At this point, the VHIP is free from constraints, and the user will fall through the surface.

The solution, as described in [50], is shown in **Fig. 5.10**. It is to iterate the process

of finding the constraining facets, and moves the VHIP to above all active facets. Once one facet is determined to be active, a temporary VHIP is calculated ($\text{VHIP}_{\text{temp}}$ in **Fig. 5.10(b)**). Then, using this new $\text{VHIP}_{\text{temp}}$ as the haptic interface point (HIP_{temp} in **Fig. 5.10(c)**), the neighboring facets are checked to see if any should be active. **Fig. 5.10(c)** shows that the line connecting the previous VHIP and the HIP_{temp} passes through the right facet within its boundaries, making it active. If there are any new constraints, a new temporary is calculated, which would be $\text{VHIP}_{\text{final}}$ in **Fig. 5.10(c)**. The maximum number of iterations is equal to the maximum number of constraints, so the iteration process is relatively quick. Once all constraints are found, In this case, there is not other constraints and the $\text{VHIP}_{\text{final}}$ becomes the new VHIP.

5.2.3 Location Computation

After all the active facets are determined, the surface contact point can be computed according to the constraints. The solution is found by minimizing the distance between the HIP and the VHIP. A common method of solving this constraint minimization problem is to use Lagrange multipliers. The analogy is to consider there is a virtual spring of unity stiffness connecting the two points with the equilibrium distance equals zero. The equation of the energy of the virtual spring is given by:

$$E = \frac{1}{2}\|\mathbf{x} - \mathbf{u}\| \tag{5.1}$$

where $\mathbf{x}$ is the coordinates of the VHIP and $\mathbf{u}$ is the coordinates of the HIP. Moreover, VHIP is constrained to be on the active facets. For each active facet, the constraint of the position of VHIP is:

$$\mathbf{N}_m\mathbf{x} = D_m \tag{5.2}$$

where $\mathbf{N}_m$ is the normal vector of the active facet. The subscript m is determined by the number of constraints which is limited by the DOF of the haptic device, ($1 \leq m \leq DOF$). **Eqn. 5.1** can now combine with **Eqn. 5.2** for all the active facets, with the Lagrange multipliers to form:

$$L = \frac{1}{2}\|\mathbf{x} - \mathbf{u}\| + \sum_{m=1}^{DOF} \lambda_m(\mathbf{N}_m\mathbf{x} - D_m) \tag{5.3}$$

where λ_m is the Lagrange multiplier coefficient for the mth constraint. The solution of the position of VHIP is the minimization of L [50].

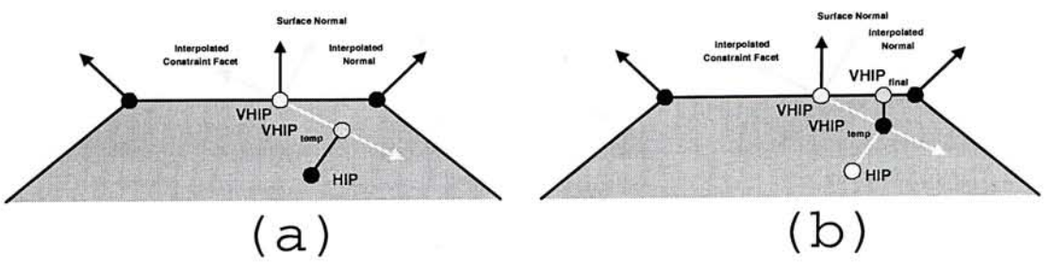


Figure 5.11: Two pass haptic shading with specified normals.

5.2.4 Force Shading

In geometric representation, a curved object is often approximated by a set of flat polygonal facets. If we compute the interaction force directly using the polygonal mesh, there will be an unrealistic distinct feeling of the feedback force across the boundary of two facets. The obvious solution is to increase the number of the polygons so that the small change of the force is imperceptible. However, increasing the number of the polygons means increasing the computational time for each haptic servo loop and thus degrades the system performance. The force shading method that we are going to discuss, uses the information found in the geometric model to allow regular polygonal facets to be perceived like a curved continuous surface.

In many graphical models, surface normals are defined at the vertices of the polygonal mesh which correspond to the surface normals of the underlying curved surface. In computer graphics, to achieve the appearance of a continuous surface, the rendering process interpolates the normal vectors (or the corresponding color values) for each pixel on the surface [7]. The lighting calculations then use the interpolated vectors instead of the geometric surface normal of the polygon. This has the effect of eliminating abrupt surface color changes at polygon boundaries without increasing the number of polygons. In computer haptics, the interpolated surface normals can also be used to give the sensation of a curved surface. In [29], the direction of the normal force is changed while the magnitude caused by the penetration of the original object is retained. In [36], an alternative approach uses a two-step procedure to determine the best VHIP position. Since this approach only alters the position of the VHIP and not directly the forces applied to the user, stable performance is much easier to guarantee.

The haptic shading method described in [36] proceeds in two passes. When the line joining previous VHIP to the HIP crosses the constraint facet within the boundaries, an additional interpolated constraint facet is defined with the interpolated normal at the intersection point. In the first pass, the interpolated constraint facet is used instead of the original active facet. The temporary $\mathrm{VHIP}_{\mathbf{temp}}$ is computed as the

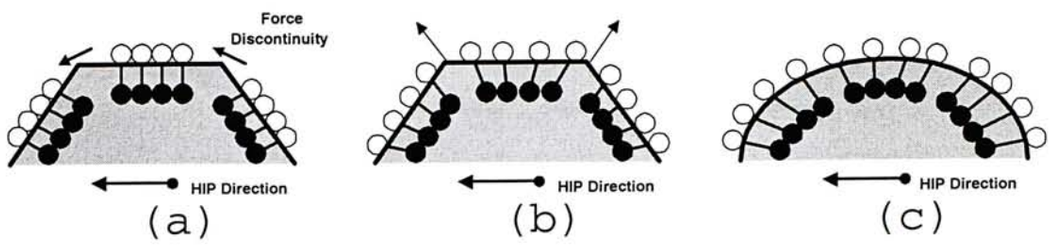


Figure 5.12: Haptic shading (b) eliminates the force discontinuities.

procedures previously discussed. This temporary position may however violate the constraints of the original polygonal geometry since it may lie above or below the true facet. Therefore in the second pass, the procedures are repeated using the original active facet. If the temporary position is below the facet as shown in **Fig. 5.11**, a valid VHIP position on the facet of the original geometry is computed. This $\text{VHIP}_{\mathbf{final}}$ is to the right of the VHIP position that would have been found if shading were not applied. Then the feedback force computed has an effect of pulling right as would be expected from an object that has surface normal illustrated. If the temporary position, after first pass, is above the true constrained facet, the $\text{VHIP}_{\mathbf{temp}}$ should be projected back onto the true constrain facet. This ensures that the $\text{VHIP}_{\mathbf{final}}$ will always be on the object surface, so that the surface effects like friction and texture will be handled correctly.

The difference between a flat surface, a haptically shaded surface and the true curved surface is illustrated in **Fig. 5.12**. The difference between the HIP and the VHIP are shown as the HIP follows a curved counter-clockwise path around the object. As seen in **Fig. 5.12(a)**, a strong discontinuity occurs when the VHIP reaches each edge of the polygonal approximation. This results in a force discontinuity, which gives the user the impression of crossing over an edge. In **Fig. 5.12(b)**, the resulting movement of the VHIP shows that the resultant force is always perpendicular to the interpolated surface just as in the case of the true curved object illustrated in **Fig. 5.12(c)**. The large instantaneous changes in force that normally occur at polygon boundaries can be minimized and the result is a surface that feels smooth and continuous.

5.2.5 Adding Surface Properties

Since there is not perfect smooth surface in the real world, several researchers [26, 25, 14] have proposed methods to simulate surface properties such as static friction, dynamic and viscous friction, stiffness and texture. These methods work by introducing

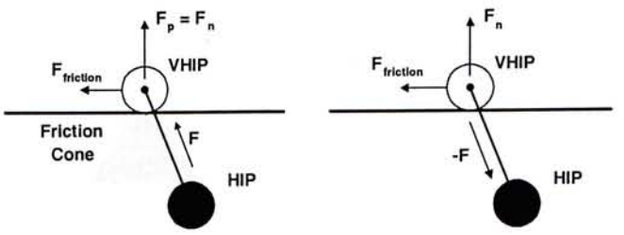


Figure 5.13: Simulation of static, dynamic and viscous friction.

additional forces to simulate the frictional forces of the contact surface. They often depend on estimating of the velocity of HIP, which makes the stability of the solution very difficult to guarantee. All these effects, however, can be created by restricting or changing the motion of the VHIP. This results in a controller which is much more stable and easier to control [36].

Static friction can be easily modeled by restricting the movement of the VHIP. In the constraint based methods, the force exerted on the VHIP by the user can be estimated by $\mathbf{F} = K(\mathbf{x} - \mathbf{u})$, where $\mathbf{x}$ is the position of the VHIP, $\mathbf{u}$ is the position of the HIP and K is the proportional gain of the haptic controller. For a given constraint plane, let $\mathbf{F}_p$ and $\mathbf{F}_t$ be the components of the force perpendicular and tangential to the constraint plane, respectively. Note that the perpendicular force is equal to $\mathbf{F}_n$, the surface constraint force. If the given constraint surface has a static friction parameter μ_s, then the VHIP is in static contact if $\|\mathbf{F}_t\| \leq \mu_s \|\mathbf{F}_n\|$, i.e., the HIP is in the friction cone of the surface. An example of such configuration is shown in **Fig. 5.13(a)**. When any constraint surface is in static contact with the VHIP, the VHIP is prevented from changing by making the new VHIP position equals to the current VHIP position.

Viscous and dynamic friction can be modeled by looking at a simplified set of equations for the motion of the VHIP. As illustrated in **Fig. 5.13(b)** the equation of the motion of the VHIP can be written as:

$$m\ddot{\mathbf{x}} + b\dot{\mathbf{x}} = -\mathbf{F} + \mathbf{F}_n - \mu_d \mathbf{F}_n \tag{5.4}$$

where $\mathbf{x}$ is the position of the VHIP, m is its mass, b is the viscous damping term and $-\mathbf{F}$, $\mathbf{F}_n$, $-\mu_d \mathbf{F_n}$ are the forces on the VHIP created by the user, the surface constraint, and the drag caused by dynamic friction respectively. Because the mass of the VHIP can be considered as being very small ($m \to 0$), it quickly reaches its saturation velocity. In dynamic equilibrium, the velocity of the VHIP is given by:

$$\dot{\mathbf{x}} = \frac{-\mathbf{F} + \mathbf{F}_n - \mu_d \mathbf{F}_n}{b} \tag{5.5}$$

This limit can be used to bound the amount that the VHIP can travel in one clock cycle. When multiple constraint surfaces exist, the lowest velocity bound is taken as the limit of the VHIP's movement. In the case that the maximum velocity is negative, the dynamic friction term is sufficient to resist all movement and the VHIP's position is not changed. If no viscous term exists, the maximum velocity is not bounded. Since this approach does not require the estimation of the HIP's velocity from a finite set of encoder values, it is not susceptible to the errors found in other approaches.

The stiffness of the surface can also be simulated by just changing the location of VHIP. Therefore the performance of the system can be optimized by keeping the controller at its optimal setting. Given a surface with stiffness s, $0 \leq s \leq 1$, it is possible to change the apparent stiffness of a surface without altering any of the controller's parameters, A new point $\mathbf{x}'$ is chosen such that $\mathbf{x}' = \mathbf{u} + s(\mathbf{x} - \mathbf{u})$, where $\mathbf{x}$ is the position of the VHIP assuming an infinitely stiff surface and $\mathbf{u}$ is the position of the HIP. The point $\mathbf{x}'$ is used as the VHIP for the computation of the feedback force. The old VHIP is still retained to allow the VHIP to continue to follow the surface of the object. A more realistic effect can be created by deformating the polygonal surface as it is affected by forces applied by the user.

Texture mapping is a well-known technique used in computer graphics to create richer and more realistic virtual environments. Texture mapping can also be applied in computer haptics to create higher fidelity scenes by using polygonal surfaces only. An image-based texture map can be used to modulate any of the surface properties described. In addition, the force-shaded constraint planes can be modified in a manner similar to *bump mapping* introduced in computer graphics [7].

Chapter 6

Haptic Rendering of Volumetric Data

Haptic rendering of the geometric models by constraint based methods allow a rich set of touch sensation of the virtual object. However, volumetric data are not comprised of geometric primitives, and thus the traditional methods developed are not directly applicable without appropriate conversion of volumetric data into geometric representation. Recently, the benefits of haptic rendering of volumetric data have been recognized, but this area of research has not yet been fully explored. Haptic interaction with volumetric data adds a new modality to volume visualization [4, 11, 19]. Visual information has an advantage in presenting whole image of the object; on the other hand, haptic information has an advantage in presenting complex attributes of local region.

There are two classes of haptic rendering methods of volumetric data, depending on the purpose of the haptic simulation. The force generated can either be constructed to convey meaningful full structural information for data exploration, or to approximate a realistic feeling of a virtual object. The first one is analogous to the direct volume rendering of volume visualization, where the reaction force is directly related to the information stored in the samples. The second one is to simulate the contact force of the isosurfaces contained in the volumetric data. Isosurface extraction is the technique used in volume visualization to explicitly represent the isosurface structure by a geometric model. The geometric model can be rendered by the geometric haptic rendering methods. The disadvantage of this approach is that the geometric model extracted by isosurface extraction methods generally contains a large number of polygons, which challenges the performance of the haptic rendering. Moreover, it requires a preprocessing of the volumetric data; thus the simulations that dynamically change

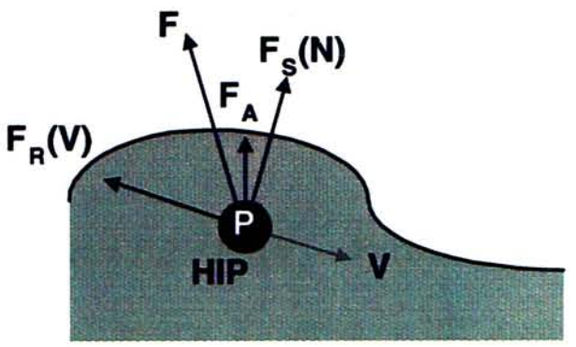


Figure 6.1: Force acting on HIP located at point **P**, which is moving at a velocity **V**.

the structure of the volumetric data are impossible. This motivates researches to develop algorithms that directly haptically render the stiff isosurfaces in the volumetric data [3, 2].

The following sections start with a discussion of the direct volume haptization methods in **Section 6.1**. Then, we present the methods of isosurface haptic rendering in **Section 6.2**. **Section 6.3** is dedicated to our proposed method for direct haptic rendering of the isosurface in volumetric data by an intermediate representation of virtual plane.

6.1 Volume Haptization

Just like direct volume rendering methods used in volume visualization, direct haptic rendering of the volumetric data has the advantage that it convey more information of the volumetric data which maybe useful for data exploration. For example, the user may wish to explore internal structures of an organ, such as lung. The method of direct haptic rendering is known as volume haptization [16], and its basic idea is to define a direct mapping of sample values to force and/or torque. The mapping are based on two principal requirements. First, the interaction force must be calculated fast enough to be used within an interactive system. Second, in order to make the visual and haptic feedback consistent, the force exerted on the user should have a direct relation with the visual appearance of the volumetric object. Therefore the force transfer function is generally defined corresponding to the transfer function of the opacity defined for visual rendering. Moreover if we employ a segmentation step to determine the visual appearance of volumetric data, we should also introduce a similar step to the haptic rendering process.

With the limitation of the haptic device to translational force only and in order to meet the speed requirement of the haptic rendering process, the interaction force is generally simplified to point contact. As the method mentioned in [2], the force that exerted on the haptic interface point (HIP) when it enters the volumetric data is specified as:

$$\mathbf{F} = \mathbf{F_A} + \mathbf{F_R}(\mathbf{V}) + \mathbf{F_S}(\mathbf{N}) \tag{6.1}$$

and is illustrated in **Fig. 6.1**. The force $\mathbf{F}$ exerted on HIP located at position $\mathbf{P}$ and moving with velocity $\mathbf{V}$ is equal to the vector sum of an ambient force $\mathbf{F_A}$, a motion retarding force $\mathbf{F_R}(\mathbf{V})$, and a stiffness force $\mathbf{F_S}(\mathbf{N})$. The ambient force is the sum of all force acting on the HIP that are independent of the volumetric data itself. It may be the resultant of some force, such as gravitational or buoyant force, which are independent of the HIP position, and synthetic force used to guide the user during interactive volume exploration. The motion retarding force is proportional to velocity and can be used to represent a viscous force. The last term captures the stiffness of the object and is always in the direction of the local gradient.

When visually rendering a volumetric data, the opacity transfer function is specified as $\alpha = t_\alpha(d, \|\mathbf{g}\|)$ [21], where the opacity value α at a point is defined by both the density (d) and the magnitude of the density gradient ($\|\mathbf{g}\|$) at that location. In order to keep visual and haptic rendering consistent, force transfer functions t_r and t_s are constructed which are similar to t_α, but produce force magnitudes rather than opacities. The retarding and stiffness force functions for haptic volume rendering become:

$$\mathbf{F_R}(\mathbf{V}) = -\mathbf{V}t_r(d, \|\mathbf{g}\|) \tag{6.2}$$

$$\mathbf{F_S}(\mathbf{N}) = \frac{\mathbf{N}}{\|\mathbf{N}\|}t_s(d, \|\mathbf{g}\|) \tag{6.3}$$

The normal vector $\mathbf{N}$ is computed using central differences. The linear correspondence between the visual transfer function and the haptic transfer functions produced an intuitive force response, as in:

$$t_r(d, \|\mathbf{g}\|) = C_1 t_\alpha(d, \|\mathbf{g}\|) + C_2 \tag{6.4}$$

$$t_s(d, \|\mathbf{g}\|) = C_3 t_\alpha(d, \|\mathbf{g}\|) \tag{6.5}$$

These force transfer functions make essentially the more opaque a material, the greater its stiffness and motion retarding properties. The stiffness function has an implied zero

additive constant to ensure that the initial contact with an object starts from a zero force. Other mappings of the opacity transfer function may be suitable depending on the type of force required. For instance, an exponential increasing opacity transfer function may be translated into a linear force response through the use of a logarithmic function.

6.2 Isosurface Haptic Rendering

A volumetric data may contain an implicit surface inside it, which is defined by an isovalue (for example, the surface of an organ in a CT-scanned medical data). In virtual surgical training and planning system, it is useful that the contact sensation of the isosurface can be haptically simulated. As motioned before, it is undesirable to apply haptic rendering to the geometric approximation generated by isosurface extraction algorithms. This motivates researchers to develop algorithms that directly haptically render the isosurfaces within the volumetric data.

In [2], Avila and Sobierajski try to calculate the stiffness and motion retarding force when interacting with volumetric isosurfaces, by relating the force transfer functions with the sampled density. The stiffness computation requires that the penetration distance of the HIP below the isosurface is available at every location in the volumetric data. While it is possible to pre-compute this distance, the technique used is to approximate the stiffness and retarding force based only on the density field. There are two reasons for this. First, if the simulation allows interactive modification of the data, creating a new distance map for the data would be prohibitive. Second, for a small penetration distance, the density field itself can give a reasonable approximation of the distance to an isosurface.

Similar to direct volume haptic rendering, the retarding and stiffness force functions used to feel a surface are dependent on transfer functions, f_r and f_s,

$$\mathbf{F_R}(\mathbf{V}) = -\mathbf{V} f_r(d) \tag{6.6}$$

$$\mathbf{F_S}(\mathbf{N}) = \frac{\mathbf{N}}{\|\mathbf{N}\|} f_s(d) \tag{6.7}$$

Here the density d is used as an indicator of penetration distance in the thin shell between the isosurface density values d_i and d_j, where $d_i < d_j$. The function $f_r(d)$ maps density values into retarding force magnitudes, while $f_s(d)$ maps density value

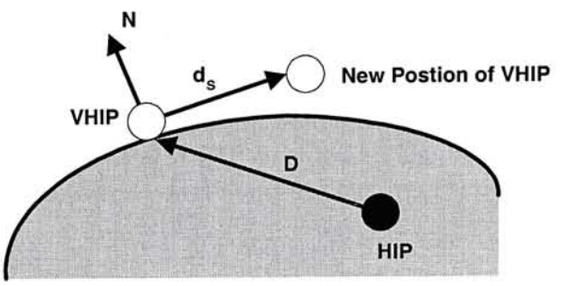


Figure 6.2: VHIP moves incrementally along the isosurface.

into stiffness force magnitudes. These functions are set to:

$$f_r(d) = \begin{cases} C_4(d - d_i) + C_5, & \text{if } d_i < d \leq d_j; \\ 0, & \text{otherwise.} \end{cases} \tag{6.8}$$

$$f_s(d) = \begin{cases} C_6(\frac{d-d_i}{d_j-d_i}), & \text{if } d_i < d \leq d_j; \\ 0, & \text{otherwise.} \end{cases} \tag{6.9}$$

The coefficients C_4, C_5, and C_6 specify a linear mapping from density values to force magnitudes. The retarding force is set to a linear function proportional to the difference in density above d_i. The stiffness force varies from zero to C_6 depending linearly on where the value d lies between d_i and d_j. This can be viewed as a penetrable shell model with viscous internal properties.

However, the distance of a point from the isosurface may not have a direct relation with the density difference. Moreover, contact sensation of some objects, such as a bone cannot be realistically simulated as a penetrable shell model. Blezek and Robb [3], try to simulate the feeling of a stiff structure by using a method which is similar to the constraint based haptic rendering methods for geometric model.

To haptically render stiff objects, the virtual point of contact with the surface (surface contact point or virtual haptic interface point, VHIP) must be maintained. Since the isosurface is implicitly contained in volumetric data, a different approach is proposed to move the VHIP along the isosurface. The VHIP is interactively moved to a new location on the surface, which follows the contours of the surface, as shown in **Fig. 6.2**. VHIP must be chosen such that the surface normal at VHIP must be equal to the vector from the HIP to VHIP, in order to approximate the motion of real objects moving across one another. In the general case, the point at which the surface normal and the vector are equal may have several solutions at disparate points on the surface. To select the correct VHIP, an *inching* algorithm is used to move the VHIP incrementally along the surface. This constrains the VHIP to the surface and prevents

Input: The current HIP, and previous VHIP.
Output: The interaction force **F**.
Algorithm: Move the VHIP incrementally, until the surface normal at VHIP is equal to the vector from HIP to VHIP, and compute the interaction force

IF HIP is outside the surface **THEN**
    VHIP $\leftarrow$ **HIP**
    Return **F** $\leftarrow$ **0**
ENDIF

VHIP $\leftarrow$ Projection of **VHIP** on the surface
N $\leftarrow$ $SurfaceNormal$(**VHIP**)
D $\leftarrow$ **VHIP** $-$ **HIP**
WHILE N $\neq$ **D DO**
    $\mathbf{d}_S \leftarrow \frac{-\mathbf{D}+(\mathbf{D}\bullet\mathbf{N})\mathbf{N}}{\|-\mathbf{D}+(\mathbf{D}\bullet\mathbf{N})\mathbf{N}\|}$
    VHIP $\leftarrow$ **VHIP** $+ \mathbf{d}_S$
    N $\leftarrow$ $SurfaceNormal$(**VHIP**)
    D $\leftarrow$ **VHIP** $-$ **HIP**
ENDWHILE

Return **F** $\leftarrow$ K(**VHIP** $-$ **HIP**)

Figure 6.3: Algorithm HR for haptic rendering of an isosurface.

ambiguities. The VHIP is moved a distance $\mathbf{d}_S$, **VHIP** $\leftarrow$ **VHIP** $+ \mathbf{d}_S$, along the surface as specified:

$$\mathbf{d}_S = \frac{-\mathbf{D} + (\mathbf{D} \bullet \mathbf{N})\mathbf{N}}{\| - \mathbf{D} + (\mathbf{D} \bullet \mathbf{N})\mathbf{N}\|} \tag{6.10}$$

where $\mathbf{D} = \mathbf{VHIP} - \mathbf{HIP}$ and $\mathbf{N}$ is the surface normal at the VHIP. After the VHIP is moved according to **Eqn. 6.10**, the new VHIP is projected onto the surface of the haptic object (to return the surface contact point). The process is repeated until $\mathbf{N} = \mathbf{D}$. The algorithm is summarized in **Fig. 6.3**:

The approach works well for synthetic quadric implicit surfaces, but is not sufficiently realistic in simulation with complex anatomic objects. The surfaces felt rough and unwanted instabilities in the haptic device are produced, especially in the high frequency regions of the volumetric data. The force artifacts are more severe in case the data has noise, which is inevitable in real-life medical data. The rough feeling of the data can be reduced by applying noise-smoothing algorithms such as low-pass filtering or morphologic closing operation. Another problem of this approach is that the surface normals are approximated rather than exactly calculated for real-life data. The algorithm occasionally fails to maintain the correct surface contact point. When

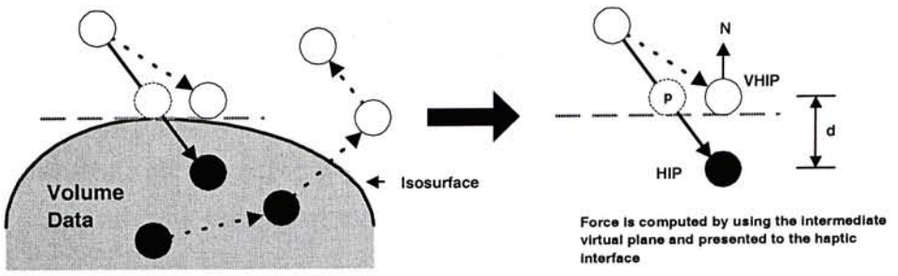


Figure 6.4: Direct haptic rendering of isosurface through an intermediate representation.

this occurs, the algorithm provides a zero length force vector, effectively rendering the object haptically transparent. Finally, the incrementally moving of the VHIP and the approximation of the surface normal by central differences may require too much computation for a servo loop of the haptic rendering for complex virtual environments. We developed a direct haptic rendering method for isosurfaces in volumetric data by an indirect geometric representation. The approach has the advantages such that the indirect geometric representation of the isosurface structure has an implicit effect of smoothing the volumetric data. Moreover, the update rate of the indirect geometric representation can be changed according to the system load and thus easily to maintain the stability of the simulation.

6.3 Intermediate Representation Approach

6.3.1 Introduction

The computation of a single haptic control loop is strongly influenced by the complexity of the object shape. For complex object, it may require a large amount of computation. Consequently, a single haptic control loop may last for too long, and a stiff surface cannot be represented. For this problem, the intermediate space has been introduced first by [1], and improved by [25]. The feature of this approach is that a virtual environment and the haptic rendering control loop exchange the necessary information by using simple geometric primitives that are frequently recomputed. Utilizing the intermediate space, it facilitates the collision detection and reaction force computation. Therefore stiff surfaces can been represented even if the virtual objects have complex shapes.

The proposed approach, as shown in **Fig. 6.4**, is to use a virtual plane as an intermediate representation of the isosurface in the volumetric data, and compute the

point interaction force based on this virtual plane. Most of the time, the simulation is not required to update the virtual plane as fast as the haptic control loop. Some servo loops will thus only use the virtual plane to generate the interaction force that can be quickly computed, as the virtual plane structure is so simple that collision detection and force computation can be done instantly. Therefore, we are able to gain higher servo rate for complex volumetric data with this method. Another advantage of this approach is that, when it is applied to a medical volumetric data, the intermediate representation of the isosurface locally by a virtual plane has an effect of smoothing the noisy volumetric data. Finally, since the volumetric data is transferred to an intermediate geometric representation, the algorithm is able to haptically simulate hybrid virtual environments that consist of both volumetric and geometric models.

In the following subsections, we will first discuss the computation of the intermediate virtual plane which captures the local information of the isosurface embedded in the volumetric data, in **Section 6.3.2**. Then, in **Section 6.3.3**, we describe how the update rate of the virtual plane depends on the underlying isosurface structure and movement of the HIP. However, the simple approach of updating the virtual plane will lead to force discontinuity artifacts. The solutions of the problems will be addressed in **Section 6.3.4**. Finally, we present the experiments and results in **Section 6.3.5**

6.3.2 Intermediate Virtual Plane

The algorithm of proposed method is summarized in **Fig. 6.5**. In each impedance control loop, the position of the HIP is traced. At the same time, the sample value v at the position of the HIP is computed by trilinear interpolation. Applying the binary segmentation function $B(v)$ to the value v, yields 0 if the HIP is located at the background (outside the defined isosurface) and 1 if it is inside the defined isosurface. A ray is constructed from the position of the VHIP (resultant position computed in pervious servo loop) to the current HIP. If two end-points of the ray are both background ($B(v) = 0$), or part of the object ($B(v) = 1$), the ray does not intersect with the isosurface. The VHIP is then allowed to move directly to the HIP and no force will be generated. Note that even when the ray is completely inside the object (both end-points have $B(v) = 1$), it is regarded as not intersecting with the isosurface and therefore no force will be generated. The reason of this is to avoid large force which exceeds the maximum capacity of the haptic device be generated, if the HIP is deeply inside the object when the simulation commences.

If the VHIP has $B(v) = 1$ and the HIP has $B(v) = 0$, the HIP is moving out of

Input: The current HIP, and previous VHIP.
Output: The interaction force **F**.
Algorithm: Update the virtual plane if necessary.
Move the VHIP according to the constraint of the virtual plane.
Compute the interaction force based on the constraint of the virtual plane.

```
IF c ≥ n THEN
    c ← 0
    IF Density(HIP) ≤ τ THEN
        VirtualPlane ← (0, 0)
        VHIP ← HIP
        Return F ← 0
    ELSE
        p ← LinearInterpolation(VHIP, HIP, τ)
        N ← SurfaceNormal(p)
        VirtualPlane ← (N, p)
    ENDIF
ELSE
    c ← c + 1
ENDIF

IF VirtualPlane ≠ (0, 0) THEN
    IF N • (HIP − P) < 0 THEN
        VHIP ← HIP + ((p − HIP) • N)N
        F ← K(VHIP − HIP)
    ELSE
        VHIP ← HIP
        Return F ← 0
    ENDIF
ELSE
    VHIP ← HIP
    Return F ← 0
ENDIF
```

Figure 6.5: Algorithm HRI for haptic rendering of an isosurface by an intermediate representation.

the object. No force should be generated for this case neither, even the end-points of the ray have different values of $B(v)$. The only case that have force output is when the VHIP has $B(v) = 0$ and the HIP has $B(v) = 1$. This case happens when the ray penetrates the defined isosurface from outside. In order to compute the interaction force in this case, a tangential plane including the intersection point on the isosurface is worked out. The virtual plane is defined by the equation:

$$\mathbf{N} \bullet (\mathbf{x} - \mathbf{p}) = 0 \tag{6.11}$$

where $\mathbf{p}$ is the intersection point computed by linear interpolation along the ray and it corresponds to the position where $v = \tau$, the density of the isosurface, and $\mathbf{N}$ is the normal at the intersection point approximated by using central differences. The computation of the virtual plane transmits the local information of the volumetric data to the intermediate space that will be used for impedance control of the haptic interface. Generally, the position and the orientation of the virtual plane should be frequently re-computed according to the movement of the HIP.

The interaction force is then computed by using the intermediate virtual plane representation of the isosurface, just like constraint based methods. The VHIP is moved to a point on the plane where the distance between it and the HIP is the minimum, computed by:

$$\mathbf{VHIP} = \mathbf{HIP} + ((\mathbf{p} - \mathbf{HIP}) \bullet \mathbf{N})\mathbf{N} \tag{6.12}$$

The reaction force is proportional to the distance between the newly computed VHIP and the HIP, approximated by the virtual spring-damper model. Force computed in this way has a direction in the normal of the surface only. Though it gives user an important sense of perception of virtual objects, we rarely experience frictionless surfaces in real life. As discussed in **Section 5.2.5**, the surface properties such as static friction, dynamic and viscous friction, stiffness and texture can be created by solely restricting the movement of the VHIP. The methods are implemented in our system to simulate the static, dynamic and viscous friction.

6.3.3 Updating Virtual Plane

If there is enough processing power, we can just set the update rate of the virtual plane to be the same as the haptic control loop. If the update rate of the virtual plane is moderately fast, the operator can feel curved surfaces. However, low update rate makes a bumpy surface, like a surface of polyhedron. Moreover, various kinds of factors also make a bumpy surface, including the curvature of the isosurface and the

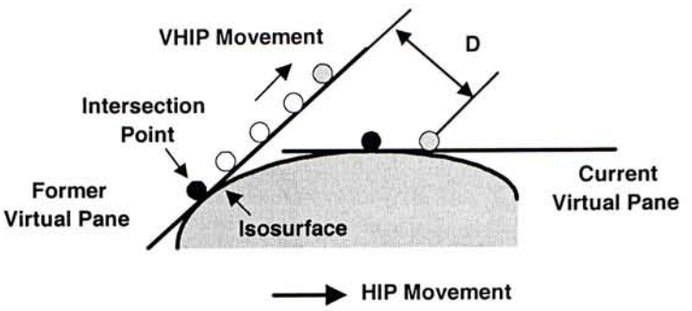


Figure 6.6: Transition of the virtual planes.

velocity of the HIP. In general, the update rate of the virtual plane should be fast enough to correctly capture the curvature of the isosurface when the HIP moves. The transition of the virtual planes is shown in **Fig. 6.6**. In order to have a smooth feeling of the surface, **D** should be as small as we cannot perceive it.

In our implementation, the update rate of the virtual plane is set to be $1/n$ of the update rate of the control loop. The parameter n can be adjusted by the user, according to curvature of the isosurface, the impedance controller parameters, and the system load. At the beginning of the simulation, the counter (c) is set to be zero and is incremented at every loop of the haptic control. When c equals n, a new virtual plane should be computed if the HIP is still inside the isosurface. The counter (c) is then reset to zero, and the reaction force is computed using the new virtual plane.

6.3.4 Preventing Force Discontinuity Artifacts

The intermediate space method works well only with the plane equation is updated frequently compared to the velocity of the HIP. As shown in **Fig. 6.7**, this will cause problem on sharply-curving surface where the update rate of the virtual plane is not fast enough to capture the information of the isosurface. A sharp discontinuity occurs in the force profile, when the HIP is allowed to move a large distance before the new virtual plane is approximated. If the previous servo loop computation leaves the VHIP outside the surface (as shown in **Fig. 6.7(a)**), the VHIP drops suddenly onto the new virtual plane. Worse, if the VHIP is embedded in the new surface (as shown in **Fig. 6.7b**), it moves violently until it leaves the surface. Since the reaction force is proportional to the distance of this movement, a force with large magnitude will be produced. The force may cause severe artifact or even exceed the bandwidth of the haptic device.

To solve the problem of the force discontinuity when the VHIP is outside the sur-

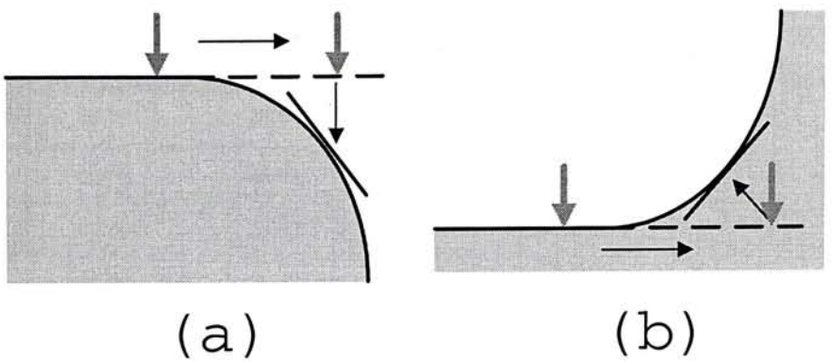


Figure 6.7: Force discontinuity results if the update rate of the virtual plan is too slow. (a) VHIP suddenly drops to the new virtual plane, (b) VHIP violently moves out the surface.

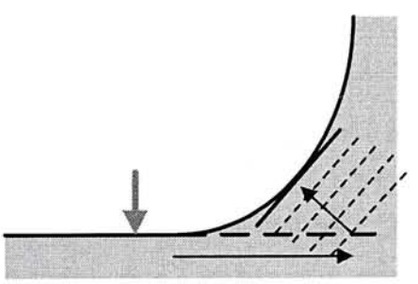


Figure 6.8: "Recovery time" solution to the problem of extreme force when the VHIP is embedded in the surface.

face, we can simply increase the update rate of the virtual plane, until the simulation is fast enough, so that the dropping distance of the VHIP is unnoticeable. Another remedy is to apply the force shading method discussed in **Section 5.2.5**. The additional constraint plane is computed using the gradient, approximated by central differences, at the VHIP of the previous servo loop. This constraint plane is first used to find a temporary VHIP, and then the true virtual plane is used to find the final VHIP. However, this method requires similar computation time for a new virtual plane at each servo loop.

To solve the problem of extreme force when the VHIP is embedded in the new surface, the *recovery time* method [25] can be used. This method is applied during the time immediately after the new virtual plane is computed. The normal direction for the force is unchanged, but the magnitude is reduced so as to bring the VHIP out of the surface over a period of time rather than instantaneously. The method is illustrated in **Fig. 6.8**. The recovery time is adjustable, and serves to move the VHIP out of the surface gradually in order to smooth the simulated interaction force.

	Intermediate Representation		Blezek and Robb [3]		Avila and Sobierajski [2]	
	Avg. Time (10^{-6} sec.)	Servo Rate (kHz)	Avg. Time (10^{-6} sec.)	Servo Rate (kHz)	Avg. Time (10^{-6} sec.)	Servo Rate (kHz)
sphere (64x64x64)	39.00	25.64	47.10	21.23	38.30	26.11
knot (64x64x64)	42.00	23.81	49.10	20.37	41.20	24.27
head (128x128x64)	48.00	20.83	59.70	16.75	47.40	21.10
lung (128x128x64)	46.80	21.37	57.60	17.36	46.20	21.65

Table 6.1: Quantitative comparison of various algorithm of direct isosurface haptic rendering (For the intermediate representation approach, the update rate of the virtual plane is set to be five times slower than the achieved servo rate).

6.3.5 Experiments and Results

GHOST SDK [15] (the development environment of the PHANToM™ haptic device), is an object-oriented toolkit that represents the haptic environment as a hierarchical collection of geometric objects and spatial effects. The isosurface haptic rendering approach was implemented by extending the GHOST SDK classes, which was written in the C++ programming language. **Table 6.1** shows the quantitative result of the proposed algorithm with the comparsion with the other two approaches. The visual rendering and haptic rendering processes are decoupled, running as separated processes which communicate by only passing the necessary information, in a SGI™ Octane®/^{MXE} R10000 workstation with 384Mb main memory.

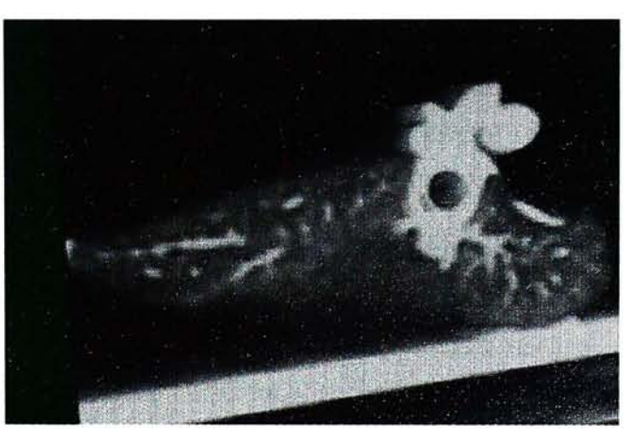
The algorithm was first tested with several sets of simple synthetic data which contain implicit surfaces. A box, a sphere and a volumetric data with three spheres of different densities were rendered both haptically and graphically. The isosurfaces rendered by our algorithm were felt correctly based on the visual image. Slower update rate caused the feeling of "step" along the surfaces, but increasing the update rate will generally improve the feeling of smoothness. We had set the update rate of the virtual plane to 5 to 50 times slower than the servo loop of the haptic control during the experiment. At most time (if the haptic interface point was not moving rapidly), the curvature of the surfaces was felt smoothly.

We then tested the algorithm with the "knot" data defined by a mathematical function. The isosurface contained in the volumetric data could be identified correctly. However, there was force discontinuities felt at concave regions if the methods discussed in the **Section 6.3.4** were not applied. At some place of high curvature, a large reaction force that even exceeded the bandwidth of the haptic device was generated. The artifact was removed by applying the solutions discussed in the **Section 6.3.4**.

Finally, we tested the algorithm with two medical volumetric data. The first one was a head model, the data was thresholded to the level of bone, and rendered both haptically and graphically. The hard structure of bone surface could be clearly felt, even we set the update rate of the virtual plane to be tenth of the haptic control loop. The other medical data is a lung model. The soft tissue the organ could be haptically located by the algorithm. It is however, the experience was not as realistic as with the skull, due to the inability of the algorithms to cope with the detailed variations on the surface. Same as other direct haptic rendering methods in the literature, the performance of our algorithm towards the surfaces of soft tissues can be improved by applying smoothing algorithms to the data (such as spatial low-pass filtering and morphological operation) before the haptic rendering process.

(a) Synthetic volumetric data.

(b) Medical volumetric data - head.



(c) Medical volumetric data - lung.

Figure 6.9: Haptic interaction with various volumetric data.

Chapter 7

Conclusions and Future Research Directions

In this chapter, we give a summary of our work and some future research directions and improvement are proposed.

7.1 Conclusions

In this thesis, we have discussed techniques for both visualization and haptic rendering of volumetric data. Firstly, we have introduced a new isosurface extraction algorithm that generates a multi-body surface from the volumetric data. Our algorithm is a generation of the skeleton climbing algorithm, which creates efficiently a usable multi-object structure by a single pass step of processing of the volumetric data. It creates precisely fit boundaries between objects without the waste of graphics resources to draw the boundaries between them twice. It is very useful when the application is to investigate a system of objects in contact, as in anatomy, geology, etc.

Then, we have presented techniques for haptic rendering of volumetric data. We have developed a direct haptic rendering method for isosurface in volumetric data using a point-based haptic feedback device. Our approach uses a virtual plane as an intermediate representation of the isosurface in the volumetric data, and computes the point interaction force based on this virtual plane. We are able to gain higher servo rate for complex volumetric data using method, as there is less computation for each control loop. Moreover, the update rate of the indirect geometric representation of the volumetric data can be changed according to the system working load and thus easily to maintain the stability of the simulation. Another advantage of this approach

when applying to the real volumetric data is that the intermediate representation of the isosurface locally by a virtual plane has an effect of smoothing the noise. Finally, since the volumetric data is transferred to an intermediate geometric representation, the algorithm is able to haptically simulated hybrid virtual environments that consist of both volume and geometric graphical models. We have tested our algorithm with synthetic and real-life volumetric data, and the experiment shows the results are promising.

7.2 Future Research Directions

View Dependent Simplification – View-dependent simplification techniques [22] have been shown useful in representing portion of the object by dense mesh while the rest of the objects are represented by coarse mesh. They have an application in speeding up the rendering while preserving the image quality. We have incorporated the adaptive skeleton climbing algorithm with our multi-body surface extraction algorithm and the result was a significant reduction in the number of triangles generated. However, ASC is a view-independent algorithm which does not used viewing information to guide the simplification process. Only geometry complexity of the enclosed isosurface is used. We believe that a view-dependent approach will further improve the algorithm in term of triangle counts. However, the algorithm will e complicated to support the feature of selective refinement. Moreover, a specially designed rendered is also needed.

Speed Up by Indexing in Span Space – In practice, there is no need for the multi-body surface extraction algorithm to process every block of samples. Since many blocks are empty, i.e. contains no isosurfaces; we can simply ignore them without performing the computational intensive skeleton climbing and volume partitioning process. This can be done in the early stage of the algorithm, when we are counting the number of types of each block. Skipping empty blocks can significantly reduce the computational time. However, it still requires classifying every sample in each block to different types. A further speedup can be achieved by indexing the blocks in span space [23] using kd-tree. By indexing the minimum and maximum values of samples within a block with block size and number of the layer where the block is located, using a 4d-tree, ASC algorithm gains an execution speed-up of up to three times. However, in our case, it is not enough for us to just locate non-empty blocks, since we need to classify blocks into empty, binary and non-binary. Then, we will skip the empty blocks, applying ASC to binary blocks, and applying our algorithm to non-binary blocks to extract a multi-body surface from the volumetric data. Therefore, we need a higher dimensional

kd-tree and more keys for indexing in span space, in order to speed up our algorithm.

Realistic Force Sensation – We have found that the integration of haptic interaction into volume visualization can lead to a better understanding of complex volumetric data. All the previously mentioned rendering techniques, though, simply generate force to feel objects, cannot revealing specific details of the data. In order to have a realistic touch sensation of the objects; we must also simulate the contact friction and texture surface properties. Research of implementing surface properties, such as friction and texture, on a haptic interface has only been a recent endeavor in the field of haptics. Friction, being relatively easy to implement, has been discussed previously. Feeling of detailed texture properties is difficult to implement, and it may require us to deal with high frequency in generated force without producing unwanted vibrations. This may include limiting the speed at user's movement and providing spherical rather than point contact. Haptic texture mapping [28, 8] has been proposed to provide natural haptic sensation in a manner similar to adding graphical texture mapping in computer graphics. However, this method suffers from the same limitations associated with graphical texture mapping, namely in medical volumetric data, patient specific textures are generally not available.

Six Degree-of-Freedom Haptic Interaction – Another limitation of discussed haptic rendering techniques is the lack of simulating rotational force. It is because the PHANToM™ device provides only three degrees of translational force feedback. An area of future research that we would like to explore is the extension of the haptic rendering algorithms for haptic devices that provide six degrees of freedom (6-DOF) in both input and output. When we use our hand or tools to do real world tasks, it inherently requires six degree-of-freedom haptics, since extended objects are free to move in three translational and three rotational directions. Affordable high-bandwidth 6-DOF haptic rendering remains an outstanding problem. It is considerably more difficult than 3-DOF point contact haptic rendering. For example, real-time collision detection requires detect all surface contact (or proximity, for a force field), instead of stopping at first evidence of it. Moreover, it requires calculating a reaction force and torque at every point or extended region of contact/proximity, and at the same time reliably maintain a 1000 Hz refresh rate. McNeely at [27], presents a simple, fast and voxel-based 6-DOF haptic rendering. Since volumetric data are voxel-based models, his algorithm can be extended to support 6-DOF haptic rendering of volumetric data.

Appendix A

Two Proofs of Multi-body Surface Extraction Algorithm

This appendix presents two proofs of the multi-body surface extraction algorithm:

1. **Occurrence of Tripoints in Negative-Positive Pairs**: In a three-type cube with tripoints, the tripoints must occur in negative and positive pairs.
2. **Validity of the General Scheme**: For the general scheme for adding tetrapoints to a messy cube, it is always possible to apply at least one of the changes in **Fig. 3.11**, until G become empty.

These two facts are very important for the construction of the complete 1-skeleton elements of the multi-body surface. Since, 1-skeleton elements on the cube surface consists of vertices (normal vertices or tripoints) and edges that connect these vertices together, it is straightforward to model the problems as graphs and present the proofs by using graph theory. Before we go to the proofs, we will first introduce some basic vocabulary and theorems of graph theory.

A.1 Graph Terminology and Theorems

In this section, we will introduce basic graph terminology and theorems that are useful for the proofs. The detailed proofs of the theorems are out of scope of this thesis, but they can be found in most textbooks of graph theory or discrete mathematics, such as [35].

Definition 1 Let V be a finite non-empty set, and let $E \subseteq V \times V$. The pair (V, E) is then called a *directed graph* (on V), where V is the set of vertices and E is its set of edges. We write $G = (V, E)$ to denote it. When there is no concern about the direction of any edge, the structure $G = (V, E)$, where E is now a set of unordered pairs from V, is called an *undirected graph.*

Definition 2 The *degree* of a vertex in an undirected graph is the number of edges incident with it, except that a loop at a vertex contributes twice to the degree of that vertex. We write $deg(v)$ to denote the degree of the vertex v.

Theorem 1 (The Handshaking Theorem) *Let $G = (V, E)$ be an undirected graph with e edges, then:*

$$2e = \sum_{v \in V} deg(v)$$

Definition 3 When (u, v) is an edge of a directed graph G, u is said to be *adjacent to* v and v is said to be *adjacent from* u. The vertex u is called the *initial* vertex of (u, v), and v is called the *terminal* vertex of (u, v).

Definition 4 In a directed graph, the *in-degree* of a vertex v, denoted by $deg^-(v)$, is the number of edges with v as their terminal vertex. The *out-degree* of v, denoted by $deg^+(v)$, is the number of edges with v as their initial vertex.

Theorem 2 *Let $G = (V, E)$ be a directed graph with e edges, then:*

$$\sum_{v \in V} deg^-(v) = \sum_{v \in V} deg^+(v) = e$$

Definition 5 A graph is called *planar* if it can be drawn in a plane without any edges crossing (where a crossing of edges is the intersection of the lines or arcs representing them at a point other than their common endpoint). Such a drawing is called a *planar representation* of the graph.

Theorem 3 (Euler's Formula) *Let G be a connected planar graph (there is a path between any two distinct vertices), with e edges and v vertices. If a planar representation of G splits the plane into r regions, including an unbounded region, then:*

$$r = e - v + 2$$

A.2 Occurrence of Tripoints in Negative-Positive Pairs

Here we want to prove that, in a three-type cube with tripoints, the tripoints must occur in negative and positive pairs. Therefore, we can join them in pairs with tri-edges of correct orientations.

First of all, let us model the 1-skeleton elements formed on the surface of a three-type cube as a directed graph $G = (V, E)$, with the normal vertices and tripoints as the vertices V and the oriented edges joining the vertices as the edges E. Tripoints can be classified as two types depending on how it separates the meeting objects. Recall that a negative tripoint has one outgoing and two incoming edges, while a positive tripoint has two outgoing edges and one incoming edge. Moreover, a normal vertex has exactly one outgoing and one incoming edge. Therefore, suppose we have m normal vertices, n_1 positive tripoints and n_2 negative tripoints, then:

$$\sum_{v \in V} deg^{-}(v) = m + n_1 + 2n_2 \tag{A.1}$$

$$\sum_{v \in V} deg^{+}(v) = m + 2n_1 + n_2 \tag{A.2}$$

By **Theorem 2**,

$$m + n_1 + 2n_2 = m + 2n_1 + n_2$$

$$\Rightarrow \quad n_1 = n_2 \tag{A.3}$$

Thus, we have proved that there must be equal number of negative and positive tripoints.

A.3 Validity of the General Scheme

Here we want to prove that, for the general scheme to add tetrapoints to a messy cube, it is always possible to apply at least one of the changes in **Fig. 3.11**, until G become empty.

First of all, note that the graph $G = (V, E)$ formed on the surface of a cube, with t vertices of tripoints and e edges. We can easily project the vertices and edges of G onto a bounding sphere of the cube. Therefore the graph G is planar (a plane is logically a sphere with infinite radius). G divides the surface of the cube (or a shrunken version of it, as we progress) into f faces, with $f \leq 8$, since every original face contains a vertex of the cube. Moreover, when we make changes to G, we never increase f. Therefore,

by **Theorem 3**,

$$f = e - t + 2 \tag{A.4}$$

Since in G every vertex is a tripoint with $deg(v) = 3$, then by **Theorem 1**,

$$\begin{aligned} & \sum_{v \in V} deg(v) = 3t = 2e \\ \Rightarrow \quad & e = \frac{3t}{2} \end{aligned} \tag{A.5}$$

By **Eqn. A.4** and **Eqn. A.5**, we have:

$$\begin{aligned} & f = 2 + \frac{t}{2} \\ \Rightarrow \quad & t = 2f - 4 \end{aligned} \tag{A.6}$$

Let us assume that there is no place where we can apply any changes in **Fig. 3.11** to G. Then every face were to meet five or more vertices, sharing each with two other faces (all vertices have $deg(v) = 3$) we have:

$$\begin{aligned} & t \geq \frac{5f}{3} \\ \Rightarrow \quad & 2f - 4 \geq \frac{5f}{3} \text{ , (by } \textbf{Eqn. A.6}\text{)} \\ \Rightarrow \quad & f \geq 12 \end{aligned} \tag{A.7}$$

Therefore, it is a contraction. Thus, there is always at least one loop with only two, three or four edges, corresponding to **Fig. 3.11(a)/(b), (d), (e)**. In other words, it means we can always apply at least one of these reduction steps until G become empty.

Appendix B

An Example of Multi-body Surface Extraction Algorithm

In this appendix, we demonstrate the multi-body surface extraction algorithm using an example of isosurfaces extraction of a messy cube. There are three steps of the process, first to build the 0-skeleton of the surface, second to build the 1-skeleton, and lastly to construct the edge loops which are then tessellated to triangles.

B.1 Step 1: Building 0-Skeleton

Suppose we are given a cube of eight samples and a set of thresholds to classify the samples into different types (as in this example, the samples are classified as 00013524), the first step of the algorithm is to find the 0-skeleton elements located at the occupied edges of the cube. As shown in **Fig. B.1**, the position of the separating vertices are computed by linear interpolation using proper thresholds.

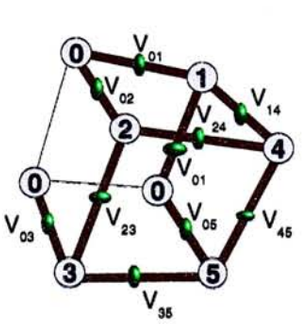


Figure B.1: Step 1: Building 0-skeleton.

B.2 Step 2: Building 1-Skeleton.

The next step of the algorithm is to find the 1-skeleton of the multi-body surface, as a set of oriented edges. There are two sub-steps, first to find the 1-skeleton elements and tripoints on the cube faces, and second to add tetrapoints and tri-edges inside the cube.

B.2.1 Step 2a: Building 1-Skeleton and Tripoints on Cube Faces

In this sub-step, we add edges joining 0-skeleton elements, and orient the edges to bound the higher type at the left. For non-binary faces, we add tripoints to separate the types. **Fig. B.2(a)** shows the 1-skeleton elements at the cube faces, and **Fig. B.2(b)** shows the flattened version with oriented edges.

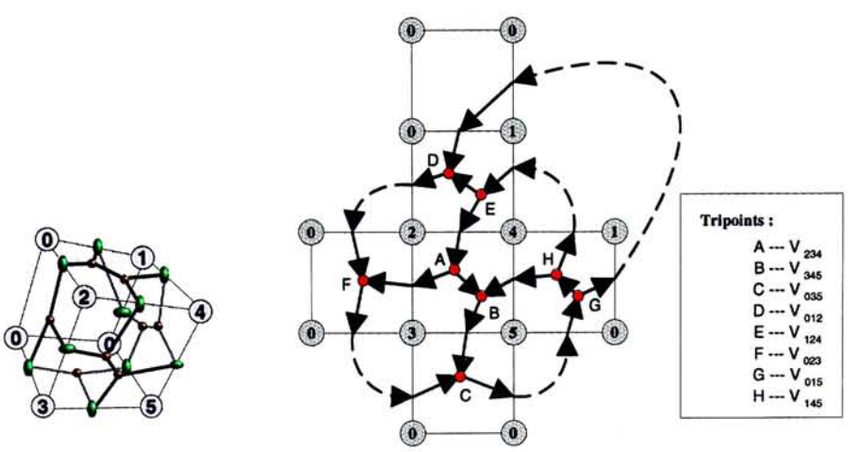


Figure B.2: Step 2a: Building 1-skeleton and tripoints on cube faces.

B.2.2 Step 2b: Adding Tetrapoints and Tri-edges inside Cube

In this sub-step, we add tetrapoints and tri-edges inside the edge, so that the complete 1-skeleton of the cube can be represented as oriented edge loops (which will be tessellated into triangles later).

First of all, according to the general scheme discussed at **Section 3.4**, we construct the initial graph G_0 which consists of the tripoints and initial edges. **Fig. B.3** shows the initial graph.

Then, we try to apply the changes in **Fig. 3.11** successively to the graph G_0 until

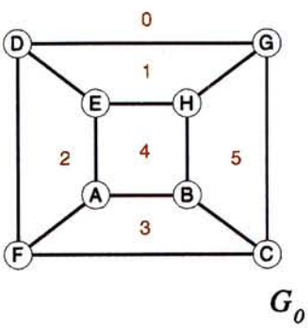
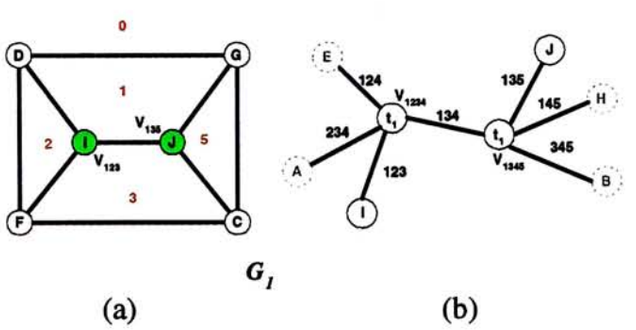


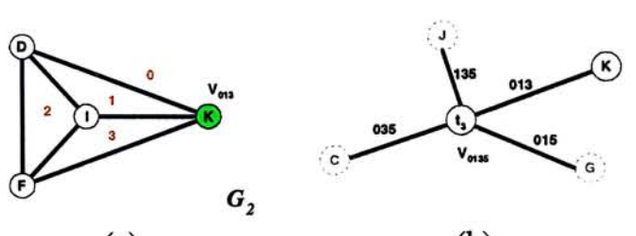
Figure B.3: Step 2b.1 : The initial graph – G_0.

it becomes empty. At the same time, we add tetrapoints, tripoints and tri-edges to the 1-skeleton. **Fig. B.4(a)** – **Fig. B.7(a)** shows the changes of G_0 to G_1, then to G_2 and G_3, until we get an empty graph G_4. The corresponding vertices and tri-edges added to 1-skeleton are shown in **Fig. B.4(b)** – **Fig. B.7(b)**.



(a) (b)

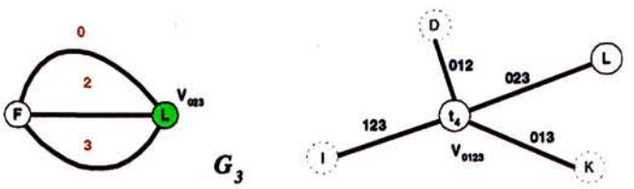
Figure B.4: Step 2b.2 : (a) Apply **Fig. 3.11(e)** to G_0 and get G_1, (b) Vertices and tri-edges added to 1-skeleton.



(a) (b)

Figure B.5: Step 2b.3 : (a) Apply **Fig. 3.11(d)** to G_1 and get G_2, (b) Vertices and tri-edges added to 1-skeleton.

After the graph reduction, we have added tetrapoints, tripoints and tri-edges to the 1-skeleton, as shown in **Fig. B.8(a)**. We find that every newly added tripoint has two tri-edges of same labels connected to them. We merge these two tri-edges and delete the tripoint, until we get **Fig. B.8(b)**.



(a) (b)

Figure B.6: Step 2b.4 : (a) Apply **Fig. 3.11(d)** to G_2 and get G_3, (b) Vertices and tri-edges added to 1-skeleton.

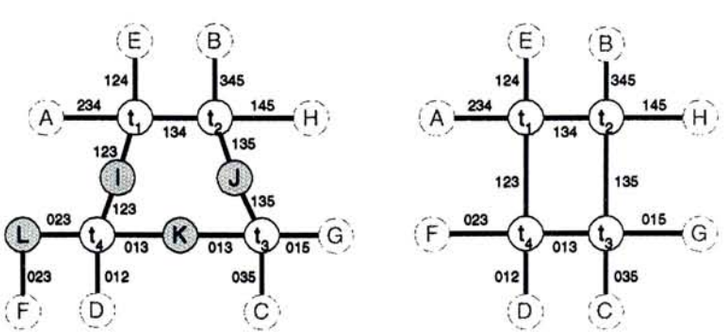
(a) (b)

Figure B.7: Step 2b.5 : (a) Apply **Fig. 3.11(a)** to G_3 and get an empty graph G_4, (b) Vertices and tri-edges added to 1-skeleton.

Then, we compute the position of the tetrapoints by minimizing the the total length of the tri-edges. It is a linear minimizing problem of solving three 4×4 matrices in x, y, z coordinates, as following:

$$\begin{pmatrix} 4 & -1 & 0 & -1 \\ -1 & 4 & -1 & 0 \\ 0 & -1 & 4 & -1 \\ -1 & 0 & -1 & 4 \end{pmatrix} \begin{pmatrix} \mathbf{t}_1 \\ \mathbf{t}_2 \\ \mathbf{t}_3 \\ \mathbf{t}_4 \end{pmatrix} = \begin{pmatrix} \mathbf{E}+\mathbf{A} \\ \mathbf{B}+\mathbf{H} \\ \mathbf{C}+\mathbf{G} \\ \mathbf{D}+\mathbf{F} \end{pmatrix} \tag{B.1}$$

Finally, **Fig. B.9** shows the complete 1-skeleton added to the cube.



(a) (b)

Figure B.8: Adding tetrapoints and tri-edges inside cube: (a) Before deletion of newly added tripoints, (b) After deletion of tripoints by merging of the tri-edges.

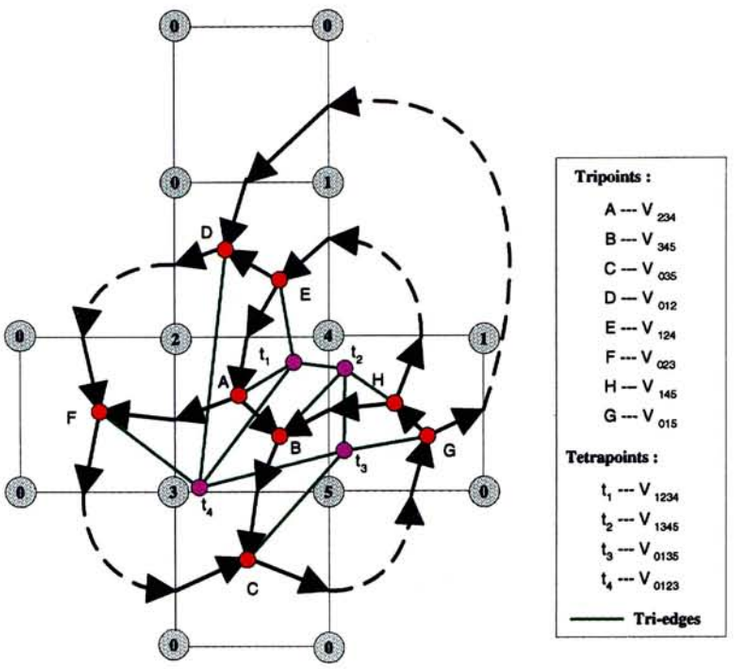


Figure B.9: Step 2: Building 1-skeleton.

B.3 Step 3: Constructing Edge Loops and Triangulating

The final step of the isosurfaces extraction is to construct the edge loops and tessellate them to triangles. By the algorithm shown in **Fig. 3.13**, we first generate the edge loops involving any types of vertices starting at every occupied edges, then we generate the edge loops involving only tripoints and tetrapoints (e.g. $\mathbf{A} \rightarrow \mathbf{B} \rightarrow \mathbf{t}_2 \rightarrow \mathbf{t}_1 \rightarrow \mathbf{A}$), and lastly we generate the edge loops involving only tetrapoints ($\mathbf{t}_1 \rightarrow \mathbf{t}_2 \rightarrow \mathbf{t}_3 \rightarrow \mathbf{t}_4 \rightarrow \mathbf{t}_1$).

After we construct all the edge loops, we then create the triangular mesh using the algorithm shown in **Fig. 3.15**.

The isosurfaces extracted are shaded in different colors and shown in **Fig. B.10**. Moreover, we also show other three examples at **Fig. B.11**, **Fig. B.12**, and **Fig. B.13**.

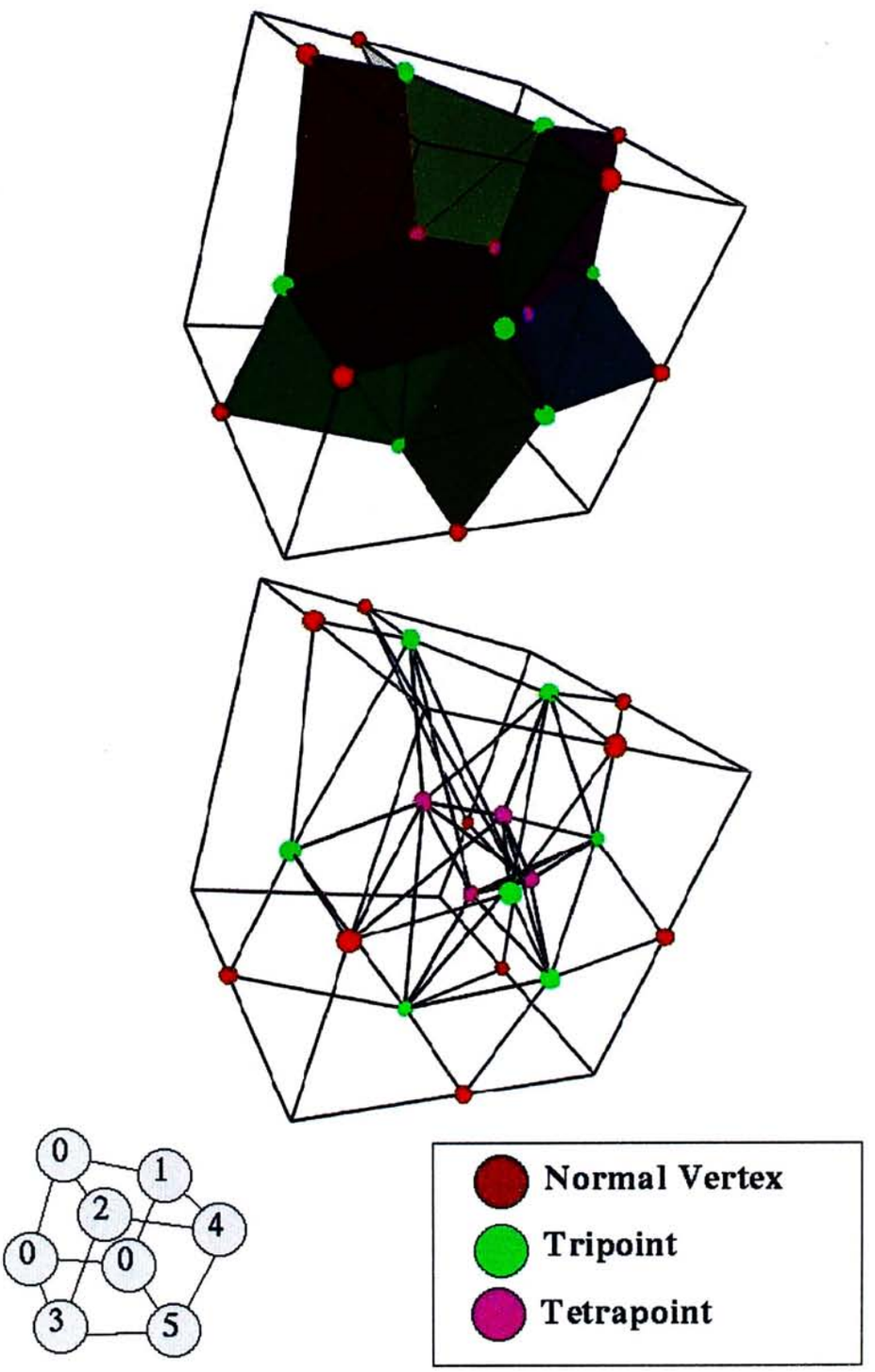


Figure B.10: Isosurfaces extracted of cube of types 00013524.

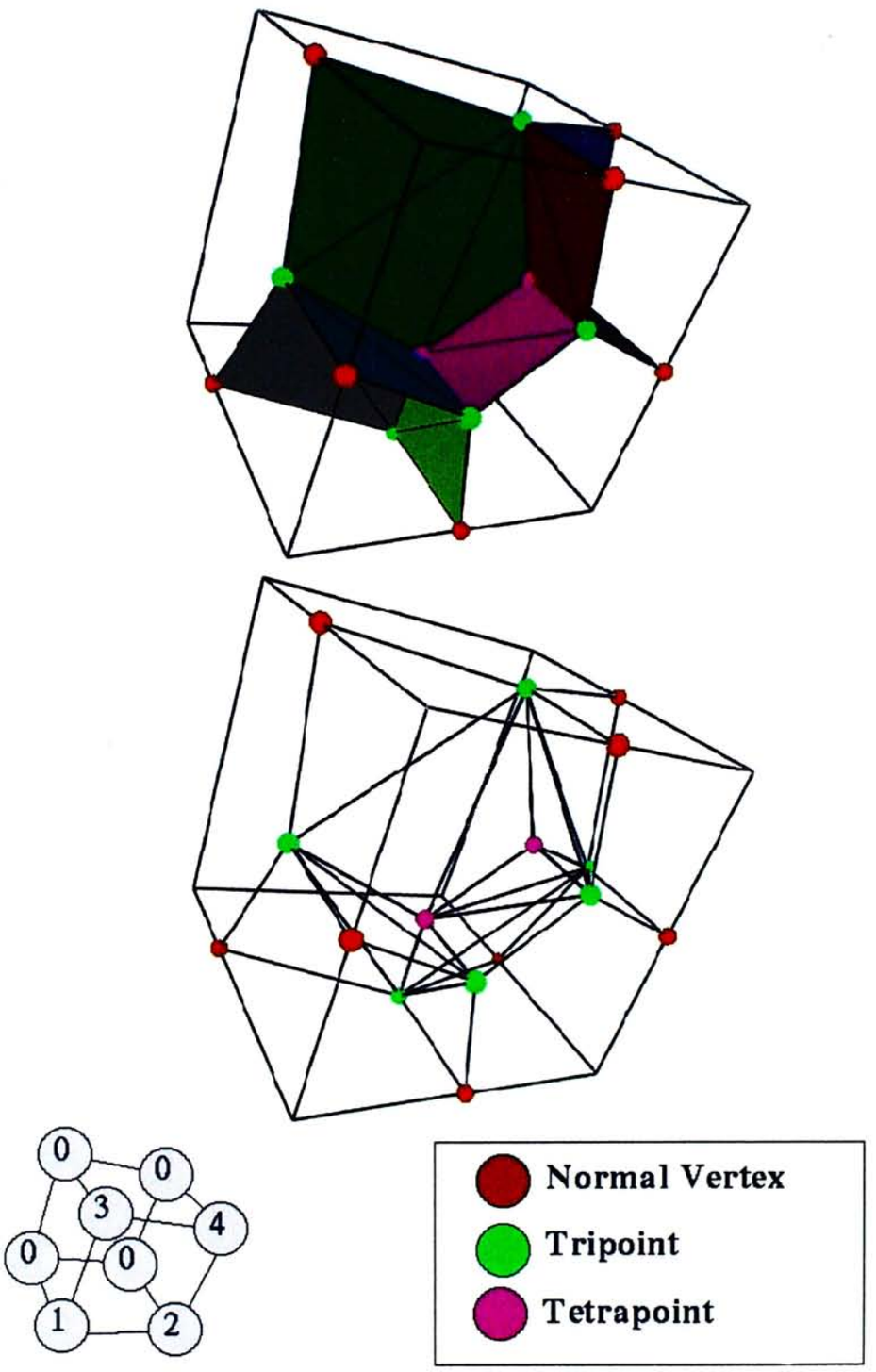


Figure B.11: Isosurfaces extracted of cube of types 00001234.

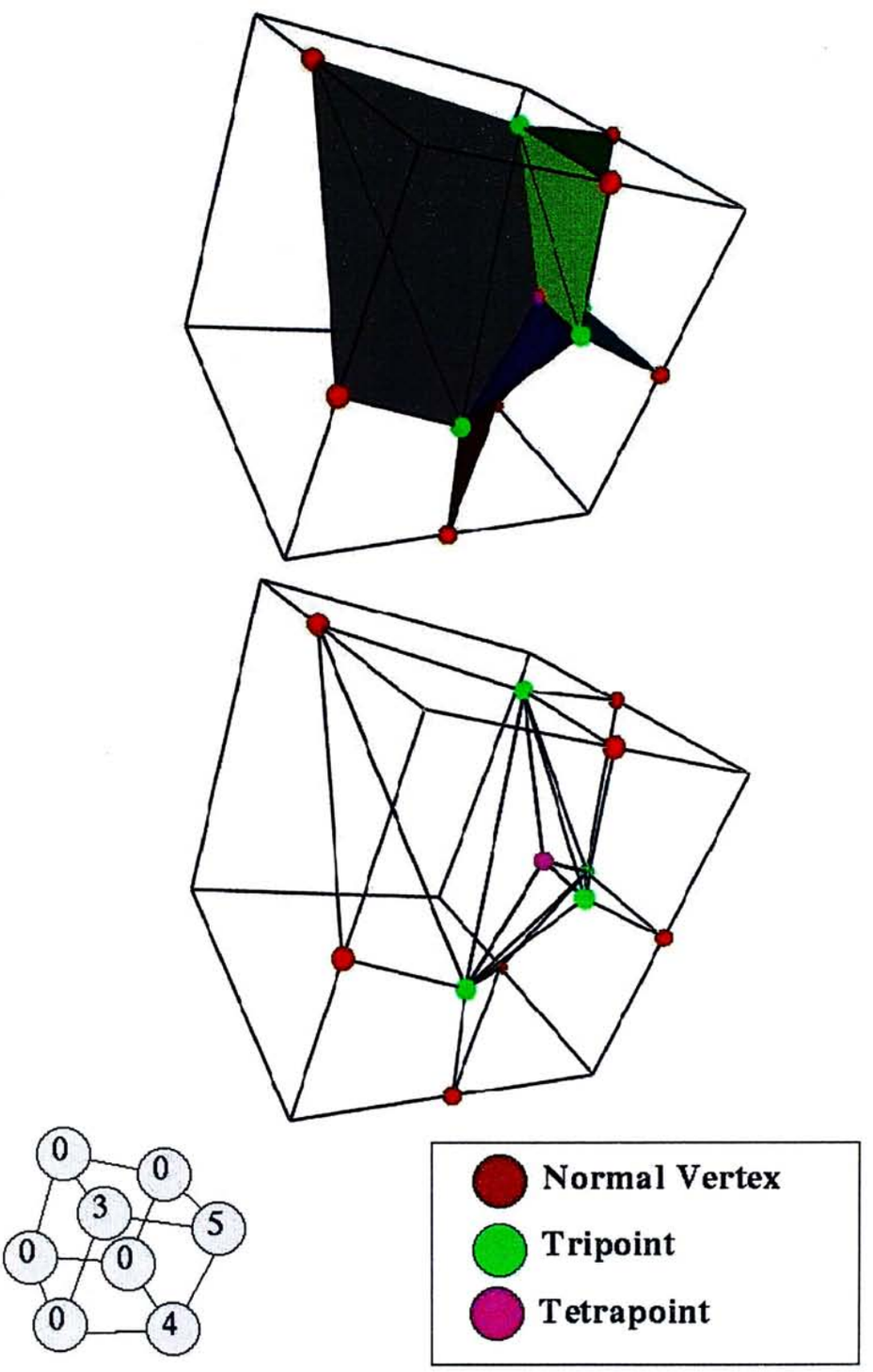


Figure B.12: Isosurfaces extracted of cube of types 00000435.

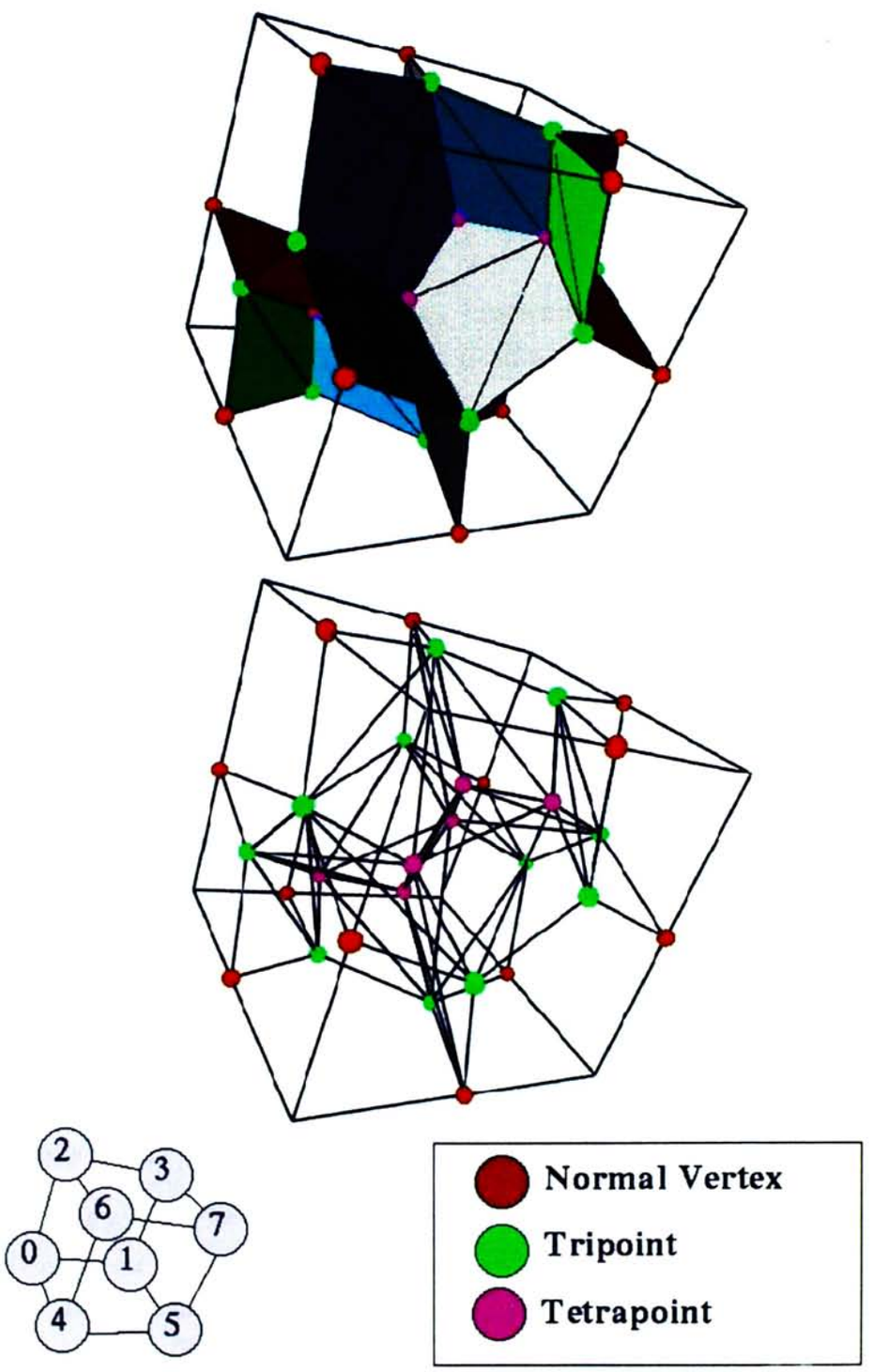


Figure B.13: Isosurfaces extracted of cube of types 01234567.

Bibliography

[1] Y. Adachi, T. Kumano, and K. Ogino. Intermediate Representation for Stiff Virtual Objects. In *Proc. IEEE Virtual Reality Annual Intl. Symp.*, pages 203–210, March 1995.

[2] Ricardo S. Avila and Lisa M. Sobierajski. A Haptic Interaction Method for Volume Visualization. *IEEE Visualization '96*, pages 197–204, October 1996. ISBN 0-89791-864-9.

[3] Daniel J. Blezek and Richard A. Robb. Haptic Rendering of Isosurface Directly from Medical Images. In *Medicine Meets Virtual Reality*, pages 67–73, 1999.

[4] F. Brooks, J. Ouh-Young, J. Batter, and P. Kilpatrick. Project GROPE – Haptic Displays for Scientific Visualization. *Computer Graphics (Proceedings of SIGGRAPH 90)*, 24(4):177–185, August 1990. ISBN 0-201-50933-4. Held in Dallas, Texas.

[5] H. Cline, W. Lorensen, S. Ludke, C. Crawford, and B. Teeter. Two Algorithms for the Reconstruction of Surfaces from Tomograms. *Medical Physics*, 15(3):320–327, June 1988.

[6] Michael J. DeHaemer and Michael J. Zyda. Simplification of Objects Rendered by Polygonal Approximations. *Computer & Graphics*, 15(2):175–184, 1991.

[7] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics Principles and Practice.* Addison-Wesley Publishing Company, Inc., 2^{nd} edition, 1996.

[8] Jason P. Fritz and Kenneth E. Barner. Stochastic Models for Haptic Texture. In *Proceedings of the SPIE International Symposium of Intelligent Systems and Advanced Manufacturing – Telemanipulator and Telepresence Technologios III*, November 1996.

[9] H. Fuchs, Z. M. Kedem, and S. P. Uselton. Optimal Surface Reconstruction from Planar Contours. *Graphics and Image Processing*, 20(10):693–702, 1977.

[10] Allen Van Gelder and Jane Wilhelms. Topological Considerations in Isosurface Generation. *ACM Transactions on Graphics*, 13(4):337–375, October 1994. ISSN 0730-0301.

[11] S. Gibson, J. Samosky, A. Mor, C. Fyock, and et al. Simulating Arthroscopic Knee Surgery Using Volumetric Object Representations, Object Representations, Real-Time Volume Rendering and Haptic Feedback. In *In First Joint Conference on Computer Vision, Virtual Reality, and Robotics in Medicine and Medical Robotics and Computer Assisted Surgery*, pages 369–378, 1997.

[12] Douglas P. Haanpaa and Gerald P. Roston. An Advanced Haptic System for Improving Man-Machine Interfaces. *Computers & Graphics*, 21(4):443–449, July 1997. ISSN 0097-8493.

[13] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh Optimization. *Proceedings of SIGGRAPH 93*, pages 19–26, August 1993. ISBN 0-201-58889-7. Held in Anaheim, California.

[14] Thomas V. Thompson II, David E. Johnson, and Elaine Cohen. Direct Haptic Rendering of Sculptured Models. *1997 Symposium on Interactive 3D Graphics*, pages 167–176, April 1997. ISBN 0-89791-884-3.

[15] SensAble Technologies Inc. *GHOST Software Developer's Toolkit*, 1997.

[16] H. Iwata and H. Noma. Volume Haptization. In *Virtual Reality*, pages 16–23. IEEE 1993 Symposium on Research Frontiers in, 1993.

[17] A. Kaufman. Volume Visualization: Principles and Advances. In *SIGGRAPH 1998 Course Notes 24 - Advances in Volume Visualization*, 1998.

[18] Philippe Lacroute and Marc Levoy. Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation. *Proceedings of SIGGRAPH 94*, pages 451–458, July 1994. ISBN 0-89791-667-0. Held in Orlando, Florida.

[19] N. Langrana, G. Burdea, K. Lange, D. Gomez, and S. Deshpande. Dynamic Force Feedback in a Virtual Knee Palpation. *Journal of Artifical Intelligence Medicine*, 6:321–333, June 1994.

[20] Noshir Langrana, Grigore Burdea, Jumoke Ladeji, and Michael Dinsmore. Human Performance Using Virtual Reality Tumor Palpation Simulation. *Computers & Graphics*, 21(4):451–458, July 1997. ISSN 0097-8493.

[21] M. Levoy. Display of Surface from Volume Data. *IEEE Computer Graphics and Applications*, 8(3):29–37, May 1988.

[22] Y. Livnat and C. Hansen. View Dependent Isosurface Extraction. *IEEE Visualization '98*, pages 175–180, October 1998. ISBN 0-8186-9176-X.

[23] Yarden Livnat, Han-Wei Shen, and Christopher R. Johnson. A Near Optimal Isosurface Extraction Algorithm Using the Span Space. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):73–84, March 1996. ISSN 1077-2626.

[24] William E. Lorensen and Harvey E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *Computer Graphics (Proceedings of SIGGRAPH 87)*, 21(4):163–169, July 1987. Held in Anaheim, California.

[25] William Mark, Scott Randolph, Mark Finch, James Van Verth, and Russell M. Taylor II. Adding Force Feedback to Graphics Systems: Issues and Solutions. *Proceedings of SIGGRAPH 96*, pages 447–452, August 1996. ISBN 0-201-94800-1. Held in New Orleans, Louisiana.

[26] Thomas H. Massie and J. K. Salisbury. The PHANToM Haptic Interface: A Device for Probing Virtual Object. In *Proceedings of the ASME Winter Annual Meeting, Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, volume 1, pages 295–301, November 1994.

[27] William A. McNeely, Kevin D. Puterbaugh, and James J. Troy. Six Degree-of-Freedom Haptic Rendering Using Voxel Sampling. *Computer Graphics*, 33(Annual Conference Series):401–408, 1999.

[28] Margaret Minsky. *Computational Haptics: The Sandpaper System for Synthesizing Texture for a Force-Feedback Display.* Phd dissertation, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 1995.

[29] H. B. Morgenbesser and M. A. Srinivasan. Force Shading for Haptic Shape Perception. In *ASME Winter Annual Meeting, Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, 1996.

[30] G. M. Nielson and B. Hamann. The Asymptotic Decider: Resolving the Ambiguity in Marching Cubes. *Visualization '91*, pages 83–91, 1991.

[31] R. Osborne, H. Pfister, H. Lauer, N. McKenzie, S. Gibson, W. Hiatt, and T. Ohkami. EM-Cube: An Architecture for Low-Cost Real-Time Volume Rendering. *1997 SIGGRAPH / Eurographics Workshop on Graphics Hardware*, pages 131–138, August 1997. ISBN 0-89791-961-0. Held in New York City, NY.

[32] H. Pfister and A. Kaufman. Cube-4 – A Scalable Architecture for Real-Time Volume Rendering. *1996 Volume Visualization Symposium*, pages 47–54, October 1996. ISBN 0-89791-741-3.

[33] Tim Poston, H.T. Nguyen, Pheng-Ann Heng, and Tien-Tsin Wong. Skeleton Climbing: Fast Isosurface with Fewer Triangles. In *Computer Graphics and Applications Proceedings*, pages 117–123, 1997.

[34] Tim Poston, Tien-Tsin Wong, and Pheng-Ann Heng. Multiresolution Isosurface Extraction with Adaptive Skeleton Climbing. *Computer Graphics Forum*, 17(3):137–148, 1998. ISSN 1067-7055.

[35] Kenneth H. Rosen. *Discrete Mathematics and Its Applications.* McGRAW-Hill, 2^{nd} edition, 1999.

[36] Diego C. Ruspini, Krasimir Kolarov, and Oussama Khatib. The Haptic Display of Complex Graphical Environments. *Proceedings of SIGGRAPH 97*, pages 345–352, August 1997. ISBN 0-89791-896-7. Held in Los Angeles, California.

[37] J. K. Salisbury, D. Brock, T. Massie, N. Swarup, and C. Zilles. Haptic Rendering: Programming Touch Interaction with Virtual Objects. *1995 Symposium on Interactive 3D Graphics*, pages 123–130, April 1995. ISBN 0-89791-736-7.

[38] J. Kenneth Salisbury and Mandayam A. Srinivasan. Phantom-Based Haptic Interaction with Virtual Objects. *IEEE Computer Graphics and Applications*, 17(5):6–10, 1997.

[39] J. Kenneth Salisbury and Mandayam A. Srinivasan. Proceedings of the First PHANToM User's Group Workshop. Technical Report AITR-1596, Massachusetts Institute of Technology, December 1996.

[40] J. Kenneth Salisbury and Mandayam A. Srinivasan. Proceedings of the Second PHANToM User's Group Workshop. Technical Report AITR-1617, Massachusetts Institute of Technology, December 1997.

[41] J. Kenneth Salisbury and Mandayam A. Srinivasan. Proceedings of the Third PHANToM User's Group Workshop. Technical Report AITR-1643, Massachusetts Institute of Technology, December 1998.

[42] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of Triangle Meshes. *Computer Graphics (Proceedings of SIGGRAPH 92)*, 26(2):65–70, July 1992. ISBN 0-201-51585-7. Held in Chicago, Illinois.

[43] R. Shekhar, E. Fayyad, R. Yagel, and J. Cornhill. Octree-Based Decimation of Marching Cubes Surfaces. *IEEE Visualization '96*, pages 335–344, October 1996. ISBN 0-89791-864-9.

[44] R. Shu, Z. Chen, and M. S. Kankanhalli. Adaptive Marching Cubes. *The Visual Computer*, 11(4):202–217, 1995. ISSN 0178-2789.

[45] M. Srinivasan and C. Basdogan. Haptics in Virtual Environments: Taxonomy, Research Status, and Challenges. *Computers & Graphics*, 21(4):393–404, July 1997. ISSN 0097-8493.

[46] Don Stredney, Gregory J. Wiet, Roni Yagel, Dennis Sessanna, Yair Kurzion, Mark Fontana, Naeem Shareef, Mike Levin, Kenneth martin, and Allison Okamura. A Comparative Analysis of Integrating Visual Representations with Haptic Display. In *Medicine Meets Virtual Reality*, pages 20–26, 1999.

[47] Herman G. T. and H. K. Liu. Three-Dimensional Display of Human Organs from Computed Tomograms. *Computer Graphics and Image Processing*, 9:1–21, 1979.

[48] Michael Teschner and Christian Henn. Texture Mapping in Technical, Scientific and Engineering Visualization. In *SIGGRAPH 1999 Course Notes 29 - Advanced Graphics Programming Techniques Using OpenGL*, 1999.

[49] Greg Turk. Re-tiling Polygonal Surfaces. *Computer Graphics (Proceedings of SIGGRAPH 92)*, 26(2):55–64, July 1992. ISBN 0-201-51585-7. Held in Chicago, Illinois.

[50] C. Zilles and J. K. Salisbury. A Constraint-Based God-Object Method for Haptic Display. In *Proceedings of the International Conference on Intelligent Robots and Systems*, volume 3, pages 146–151, August 1995.